July, 2007

Advisor Answers

Adding standard comments

VFP 9/8

Q: I'm trying to discipline myself to always add comments in certain situations, such as a header comment in every program and a change comment when I modify existing code. Does VFP offer anything to make this easier and to help me do it the same way every time?

A: This being FoxPro, there's not just one way to do what you want, but several. I'll show you two ways. One of them is best for those who like to do as much as possible from the keyboard, while the other is better for mouse-o-philes.

The mouse-heavy technique is easier, so we'll start there. It uses the Toolbox, which was added in VFP 8. One of the built-in categories in the Toolbox is Text Scraps. Text Scraps are lines you can simply drag and drop into any code window. While that's handy for any bit of code you need to use repeatedly, the technique is really much more powerful than a simple drag-and-drop repository because you can use textmerge to create the actual text on the fly.

I'll demonstrate by showing how you can add a standard change comment to the Toolbox. Here's what the comment should look like when dropped into a block of code:

```
*************************************
* Modified 7-June-2007 by TEG
*
```

Of course, the actual date should appear, not the date you created the comment template.

To add such a text scrap, open the Toolbox (Tools > Toolbox or use the icon on the standard toolbar) and expand the Text Scraps category. Right-click and choose Customize Toolbox to open the dialog shown in Figure 1. Because the Text Scraps category was expanded, the dialog opens pointing to that category. If a different category is highlighted, click on Text Scraps in the left pane.

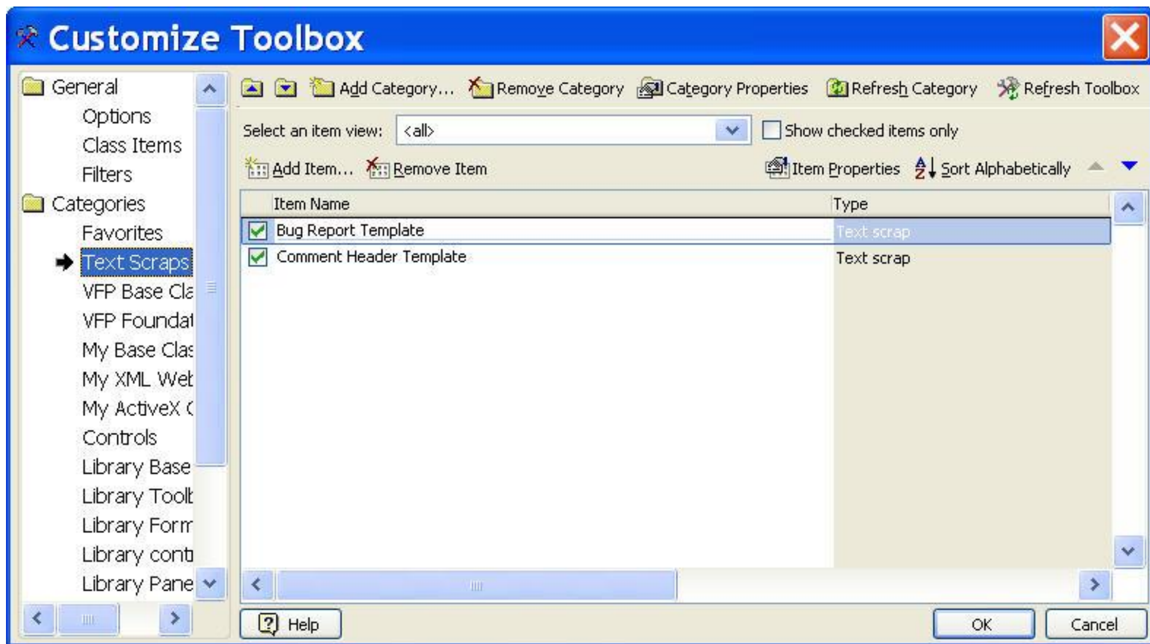

Figure 1. Adding standard comments to the Toolbox. Most toolbox items are added through the Customize Toolbox dialog.

Click the Add Item button just above the list in the right pane. The Add Item dialog (Figure 2) opens. Because the Text Scraps category was chosen, the dialog defaults to adding a text scrap. After you click OK, the Item Properties dialog (Figure 3) opens. This is where you actually specify the item.

Figure 2. Adding text scraps. This dialog determines the type of item to add to the Toolbox.

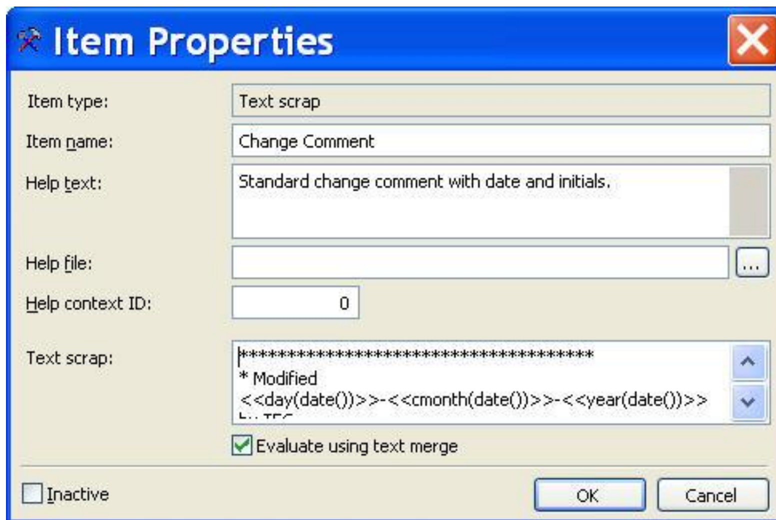

Figure 3. Specifying a change comment. Use the text scrap section of the Item Properties dialog to specify the actual text to insert.

The Item name is what appears in the Toolbox. Put the information you want to appear in the Help panel of the Toolbox in Help text. In the Text scrap editbox, put the text you want to drag and drop. If you use textmerge, check the Evaluate using text merge checkbox below. For the change comment, the editbox contains:

```
*************************************
* Modified <<day(date())>>-<<cmonth(date())>>-<<year(date())>> by TEG
*
```

There are a couple of items to note here. First, I like the date in the format shown in the original example. The complex-looking expression here builds that format. In addition, though you can't see it here, there's no return after the comment indicator on the third line. So when I drop this scrap, the cursor will be left on that line, ready for me to type the description of the change I'm making.

Once the item is created in the Toolbox, using it is easy. Just drag it from the Toolbox and drop it wherever you want the scrap to appear.

Although creating a text scrap like this is pretty easy and using it is as well, I'm really more of a keyboard-oriented person. I'd rather not remove my hands from the keyboard. So my preference for standard comments is an IntelliSense script. Scripts are a little harder to write, but I find them easier to use.

To create an IntelliSense script, open the IntelliSense Manager (Tools > IntelliSense Manager) and click on the Custom tab (shown in Figure 4).

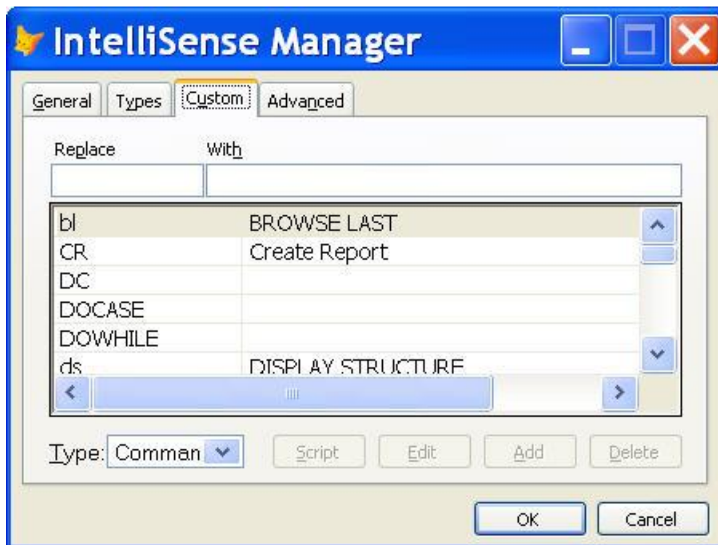

Figure 4. Adding IntelliSense scripts. Use the Custom page of the IntelliSense Manager to add your own IntelliSense items.

To add a new script, type the trigger string in the Replace textbox, then click the Add button. The trigger string is the string you'll type to run the script. For my change comment, I use TEGMOD as the trigger string.

Once you've added the item to the list, find and highlight it and then click Script to open a window that lets you create the script. IntelliSense scripts receive a single parameter, an object reference to the IntelliSense engine. Here's the code for my change comment script:

```
lparameters toFoxCode
local lcReturn

if toFoxCode.Location <> 0
  toFoxCode.ValueType = 'V'
  lcReturn = GetText()
endif toFoxCode.Location <> 0
return lcReturn

function GetText
local lcText, nDay, cMonthName, nYear, dToday
dToday = date()
nDay = day(dToday)
cMonthName = cmonth(dToday)
nYear = year(dToday)
text to lcText textmerge noshow
*************************************
* Modified <<nDay>>-<<cMonthName>>-<<nYear>> by TEG
* ~
endtext
```

```
return lcText
```

The IF statement indicates that this script shouldn't run in the Command Window. In any other code window, replace the trigger string with the return value of the function GetText. GetText puts together the same string as the text scrap does, using textmerge to insert the date. The tilde character (~) specifies the cursor location after the insertion.

I created this script by looking at other scripts and figuring out what I needed. In addition to the various scripts that come with VFP, you'll find a nice selection at http://fox.wikis.com/wc.dll?Wiki~IntelliSenseCustomScripts~VFP.

To use my script, I type the trigger string (TEGMOD) and press space or enter. The comment appears with the cursor in just the right place for typing the explanation. I find that having this script significantly increases the odds of my actually adding such a comment.

–Tamar

Getting into the Debugger

VFP 9/8/7

Q: I'm trying to use the Debugger more, but find it hard to know where to put SET STEP ON in my code to help me find problems. Is there a better way to stop code execution?

A: There are a number of better ways to get access to running code than SET STEP ON. (In fact, I can't remember the last time I used SET STEP ON or its alter-ego, SUSPEND.) The key to stopping code so you can step through it is breakpoints. A breakpoint stops execution and lets you take a look at what's going on.

Breakpoints come in two basic flavors, those based on where you are in the code and those based on the value of an expression. VFP lets you set both kinds, as well as a hybrid of the two, and provides several ways to set each.

The most obvious way to set breakpoints is by using the Breakpoints dialog (figure 6), available from both the VFP menu and the Debugger. The Type dropdown lets you choose the kind of breakpoint to create. There are four choices:

- Break at location—stop execution before running the specified line;
- Break at location if expression is true—stop execution before running the specified line, but only if the specified expression evaluates to True.
- Break when expression is true—stop execution at the end of the line in which the specified expression becomes True.
- Break when expression changes—stop execution at the end of the line in which the value of the specified expression changes.

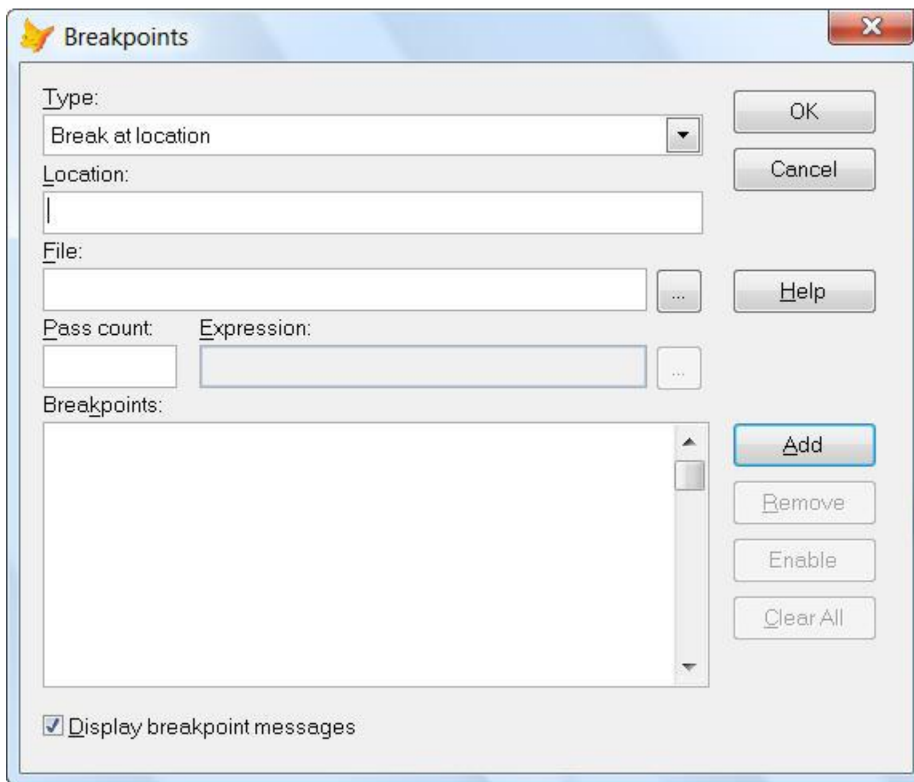

Figure 6. The Breakpoints dialog. You can set breakpoints using this dialog; breakpoints you set in other windows show up here, as well.

Be aware that the first two kinds of breakpoints (the line-based breakpoints) stop *before* executing the specified line, while the last two (the expression-based breakpoints) stop *after* executing the line that changes the expression.

To create a breakpoint using the dialog, choose the type and then enter the additional information needed for that kind of breakpoint. Be sure to hit the Add button after specifying all the information to transfer the breakpoint to the list at the bottom of the dialog.

While you can create any breakpoints you need in the dialog, VFP offers a number of other, more convenient, ways to create them, both in the editor and in the Debugger.

In any code editing window (that actually contains some code), you can set "Break at location" breakpoints three ways; all of them require you to have the selection margin (the gray bar at the left side of the window) turned on. The simplest way to set a breakpoint is to double-click in the selection margin next to the line where you want to stop. You can also choose Toggle Breakpoint from the editor's shortcut menu (shown in Figure 7). If you prefer the keyboard, you can press the F9 key.

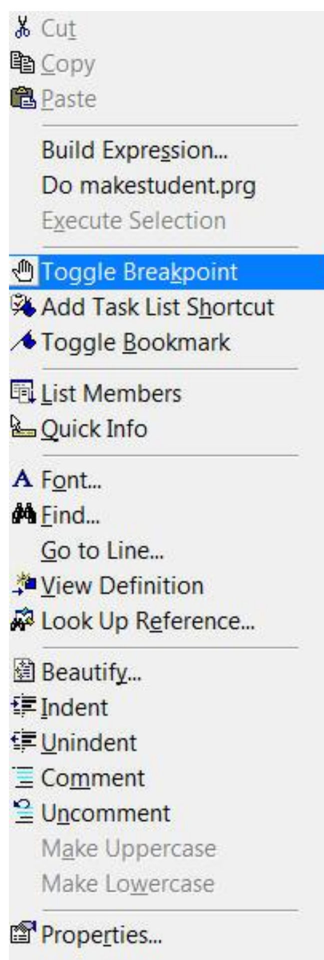

Figure 7. Editor's context menu. You can turn breakpoints on and off using the shortcut menu available in code editor windows.

However you set a breakpoint, a red dot appears in the selection margin, as in Figure 8. In addition, if it's not already open, the Debugger opens; that's because breakpoints only fire when the

Debugger is open. Like the context menu item, double-clicking is a toggle, so you can double-click on a breakpoint in the selection margin to remove it.

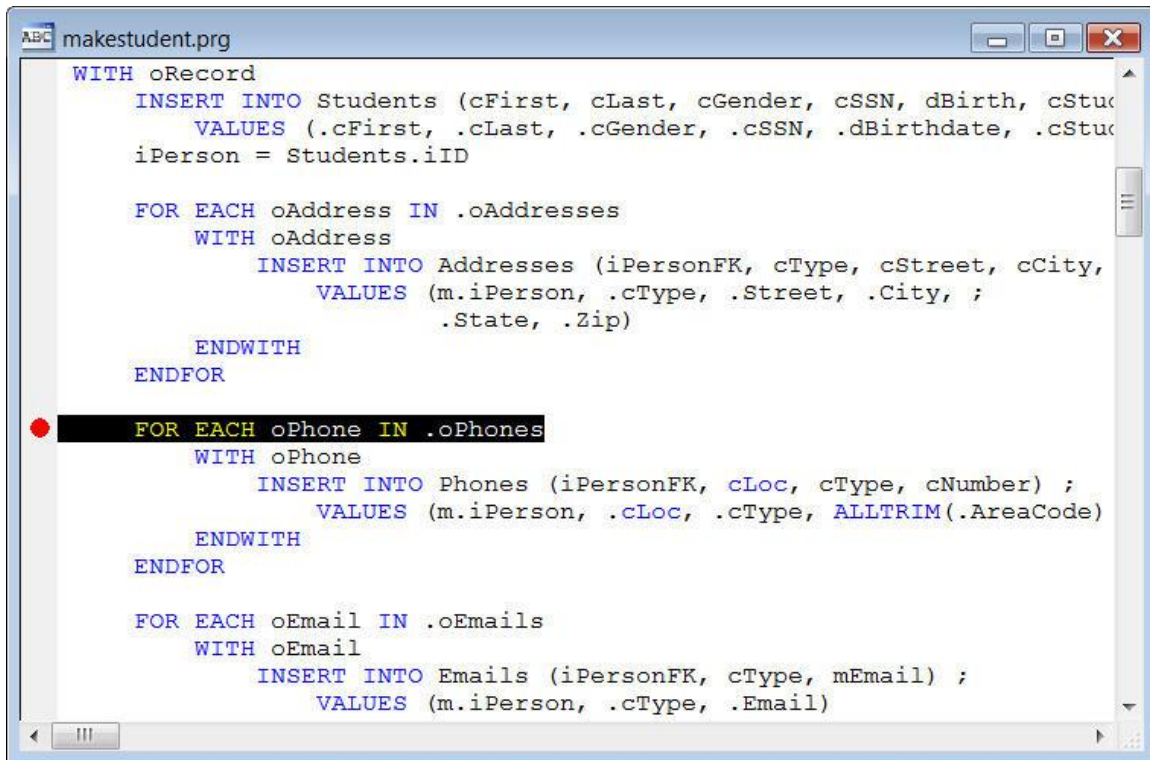

Figure 8. Breakpoint in editor window. A red dot in the selection margin indicates there's a breakpoint on a line. The dot appears however you create the breakpoint.

The Debugger offers several other ways to set breakpoints. Like code editing windows, the Trace window has a selection margin; double-click there to set or remove breakpoints (figure 9). Breakpoints set elsewhere show up in the Trace window, as well.

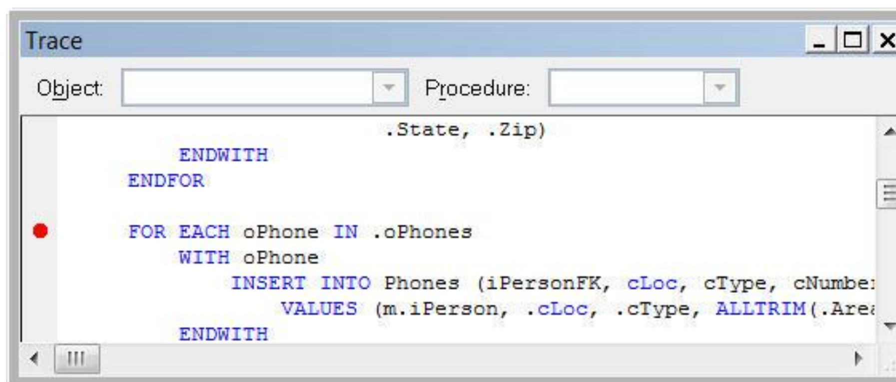

Figure 9. Setting breakpoints in Trace. Double-click in the selection margin to set a "Break at location" breakpoint in the Trace window.

Breakpoints set in code editing windows and in the Trace window are all "Break at location" breakpoints. The Watch window lets you create "Break when expression has changed" breakpoints. Double-click in the selection margin next to any item in the Watch window to set a breakpoint that fires when the value of that item changes. Figure 10 shows a breakpoint that lets you stop execution when a particular table is opened or closed. (Technically, it stops when the alias opens or closes, of course.)

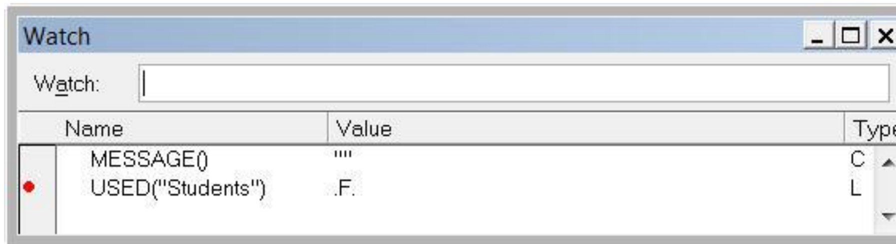

Figure 10. Breakpoint in Watch window. You can set breakpoints on expressions in the Watch window. They're the "Break when expression has changed" type.

Once you know how to set breakpoints, the question is what breakpoints to set. If you think your code is going wrong in a particular section, set a breakpoint at the beginning of that section. One feature I tend to forget to use is the Pass Count setting for "Break at location" breakpoints; rather than stopping the first time a particular line is executed, you can specify how many times it should run before stopping. So if you have a process that's working the first 255 times, but failing on pass 256, you can set a breakpoint with pass count set to 256 (or perhaps, 255, so you can see it work as it should once) and then run the code, knowing it will stop before the problem occurs.

There are a number of expressions that provide useful breakpoints. If a variable is changing unexpectedly, setting a breakpoint on it in the Watch window will let you find out what's changing it (but see the warning below for a bit of a trap). There are lots of other useful choices. Here are a few I use:

- LINENO()—stops execution on the next line of code to execute. Handy for breaking into code when READ EVENTS is in effect.
- "XXX"$PROGRAM()—fill in the name (or partial name) of a method or function you want to trace and code stops as soon as you get there. This is especially useful for getting inside forms and classes.
- USED("XXX")—fill in the alias of a table to stop when it's opened or closed.

- RECCOUNT("XXX")—fill in the alias of a table to stop when a record is added.
- DELETED("XXX")—fill in the alias of a table to stop when the current record is deleted.
- RECNO("XXX")—fill in the alias of table to stop when the record pointer moves.
- SET("XXX")—fill in one of VFP's settings to stop when it changes.

The one area where Watch window breakpoints don't work as well as you'd like is, in fact, stopping when a variable changes. The problem is that the breakpoint also fires whenever the variable goes in or out of scope. So tracing a variable local to a particular method or function can be a little tedious. If you want to watch for a particular value, you can use an expression like:

```
TYPE("MyVar") = "U" OR MyVar=100
```

This breakpoint fires when MyVar is set to 100. However, there's no good way to simply watch for a variable to change without falling into the scope trap. The same problem applies to any breakpoint that includes an alias when the data session changes.

A few final notes. In editing windows, and the Trace and Watch windows, you can only add and remove breakpoints. The Breakpoints dialog also lets you leave a breakpoint defined, but turn it off. Just uncheck it in the list at the bottom of the dialog.

The original release of VFP 7 had a bug that slowed execution if any breakpoints were defined, whether or not the Debugger was open. This was fixed in VFP 7 SP1 and hasn't reappeared in later versions.

Breakpoints are a really powerful tool. Together with the VFP Debugger's ability to step through code, they make figuring out where code is going wrong much easier than earlier techniques. Once you get used to working with breakpoints, you'll probably find you don't have much use for SET STEP ON any more.

–Tamar