January, 2007

## Advisor Answers

## Rounding to the nearest anything

VFP 9/8/7

Q: I know I can use VFP's ROUND() function to reduce the number of decimal places in a numeric value, but what if I need to round to the nearest 100, or the nearest multiple of 5?

A: There's a two-part answer to your question. First, VFP's ROUND() function is pretty versatile. It can not only round to a number of decimal places, but it can actually round to a number of places to the left of the decimal. The secret is in the second parameter, which is a little strange.

Pass a positive number as the second parameter to ROUND() to handle decimal places. For example:

```
? ROUND(37.4738, 2)   && returns 37.47
? ROUND(3827.1689, 3) && returns 3827.169
```

Pass 0 to round to the nearest integer. For example:

```
? ROUND(37.4738, 0) && returns 37
? ROUND(37.68, 0)   && returns 38
```

Note that ROUND() with a second parameter of 0 is not necessarily the same as using INT(). INT() removes decimal places with no rounding. For example:

```
? INT(37.68) && returns 37
```

Finally, pass a negative number to round to positive powers of 10. For example:

```
? ROUND(37.4738, -1) && returns 40
? ROUND(3827, -2)    && returns 3800
```

So if the number you want to round to is a power of 10, all you need is ROUND().

Be aware that ROUND() always rounds up on a 5, which can make a difference for statistical work. For a discussion of rounding algorithms, check out http://en.wikipedia.org/wiki/Rounding. Also, be aware that local customs may differ regarding the right way to round.

If you want to round to anything other than a power of 10, you have to write your own code. Fortunately, it's pretty easy to do so. Here's my function called RoundToNearest. You pass the number you want to round and the value whose nearest multiple you want to land on (call that the rounding value).

```
LPARAMETERS nValue, nNearest

LOCAL nRemainder, nReturn

nRemainder = MOD(m.nValue, m.nNearest)
IF m.nRemainder >= m.nNearest/2
   nReturn = m.nValue + (m.nNearest - m.nRemainder)
ELSE
   nReturn = m.nValue - m.nRemainder
ENDIF

RETURN m.nReturn
```

The function calls on MOD(), which returns the remainder when its first parameter is divided by its second. Here, it's used to see how far the specified number is from the next multiple of the rounding value. If we're halfway there or more, round up; otherwise, round down.

To use RoundToNearest, just pass the two parameters:

```
? RoundToNearest(37, 5)       && returns 35
? RoundToNearest(38567, 300)  && returns 38700
```

RoundToNearest.PRG is included on this month's Professional Resource CD.

–Tamar

## Disappearing Controls

VFP 9/8

Q: I have an application that was originally written in FoxPro 2.x. We've been running it in VFP 6 for some time now. I just upgraded to VFP 9 and suddenly the forms are acting very strange. As soon as the mouse enters the form, all the controls disappear. They reappear temporarily when the mouse passes over them and only reappear to stay when the user tabs through them. What's going on?

A: VFP 8 and VFP 9 support Windows themes. However, FoxPro 2.x-style forms don't. They're simply incompatible. The quick solution is to turn themes off in your application. In your start-up code, issue:

```
_SCREEN.Themes = .F.
```

Of course, for applications that are still being modified and extended, the long term solution is to recreate the forms in VFP. That has the advantage of not only supporting themes, but of giving you much better-looking forms.

–Tamar

## Hidden Variables

VFP 9/8/7

Q: My error handler includes a memory listing collected with LIST MEMORY. I noticed that sometimes a variable shows its scope as "(hid)" rather than "Pub", "Priv" or "Local". What does that mean?

–John McDonald (via Advisor.COM)

A: As you can probably guess, "(hid)" stands for hidden. As far as I can tell, there's only one way to hide a variable in VFP. That's by declaring a variable private in a procedure or function that already exists as a private variable in another routine higher in the calling chain.

Here's some code to demonstrate:

```
LOCAL cLocal, cLocalParm
PRIVATE cPrivate, cPrivateParm, cPrivateInMain
PUBLIC cPublic, cPublicParm

STORE "abc" TO cLocal, cLocalParm
STORE "def" TO cPrivate, cPrivateParm, cPrivateInMain
STORE "ghi" TO cPublic, cPublicParm

STRTOFILE("In main routine, before calling subproc" + ;
          CHR(13) + CHR(10), "TestListMemo.txt")
STRTOFILE("************************************" + ;
          CHR(13) + CHR(10), "TestListMemo.txt", .t.)
LIST MEMORY TO FILE TestListMemo.txt ADDITIVE NOCONSOLE
STRTOFILE(CHR(13)  + CHR(10), "TestListMemo.txt", .t.)

Subproc(cLocalParm, cPrivateParm, cPublicParm)

STRTOFILE("In main routine, after calling subproc" + ;
          CHR(13) + CHR(10), "TestListMemo.txt", .t.)
STRTOFILE("************************************" + ;
          CHR(13) + CHR(10), "TestListMemo.txt", .t.)
LIST MEMORY TO FILE TestListMemo.txt ADDITIVE NOCONSOLE
STRTOFILE(CHR(13)  + CHR(10), "TestListMemo.txt", .t.)
```

```
RETURN

PROCEDURE SubProc
LPARAMETERS cParm1, cParm2, cParm3

LOCAL cLocal, cPrivateInMain
PRIVATE cPrivate
PUBLIC cPublic

cLocal = "local from sub"
cPrivate = "private from sub"
cPublic = "public from sub"
cPrivateInMain = "private from main, local here"

STRTOFILE("In subproc", "TestListMemo.txt" + ;
          CHR(13) + CHR(10), .t.)
STRTOFILE("************************************" + ;
          CHR(13) + CHR(10), "TestListMemo.txt", .t.)

LIST MEMORY TO FILE TestListMemo.TXT ADDITIVE NOCONSOLE
STRTOFILE(CHR(13)  + CHR(10), "TestListMemo.txt", .t.)

RETURN
```

This code creates a text file showing the contents of memory at three points in the process: before calling the procedure, inside the procedure and after returning from the procedure. Here's the important part of the output before the procedure:

```
CPUBLIC     Pub    C  "ghi"
CPUBLICPARM
            Pub    C  "ghi"
CLOCAL      Local  C  "abc"  testlistmemo
CLOCALPARM  Local  C  "abc"  testlistmemo
CPRIVATE    Priv   C  "def"  testlistmemo
CPRIVATEPARM
            Priv   C  "def"  testlistmemo
CPRIVATEINMAIN
            Priv   C  "def"  testlistmemo
```

There are no surprises here. Each of the variables has the value assigned in the code and shows with the specified scope. Inside the procedure, things are a little more interesting:

```
In subproc
****************************************

CPUBLIC     Pub    C  "public from sub"
CPUBLICPARM
            Pub    C  "ghi"
CLOCAL      Local  C  "abc"  testlistmemo
CLOCALPARM  Local  C  "abc"  testlistmemo
```

```
CPRIVATE    (hid)  C  "def"  testlistmemo
CPRIVATEPARM
            Priv   C  "def"  testlistmemo
CPRIVATEINMAIN
            Priv   C  "def"  testlistmemo
CPARM1      Local  C  "abc"  subproc
CPARM2      Local  C  "def"  subproc
CPARM3      Local  C  "ghi"  subproc
CLOCAL      Local  C  "local from sub"  subproc
CPRIVATEINMAIN
            Local  C  "private from main, local here"  subproc
CPRIVATE    Priv   C  "private from sub"  subprocroc
```

There are now two instances of cLocal, cPrivate and cPrivateInMain, one for the main program and one for the procedure. Note also that cPrivate in TestListMemo is shown as hidden. That's because the declaration of cPrivate in SubProc prevents that procedure from seeing TestListMemo's instance of cPrivate. However, cLocal isn't shown as hidden in the main program because there's no expectation that it would be available in the procedure.

Interestingly, cPrivateInMain, which is a private variable in the main program, but a local in the procedure, isn't shown as hidden, although the variable from the main program is unavailable in the procedure. I suspect that the mechanism for displaying a variable as hidden wasn't updated with the addition of local variables.

Finally, after returning from the procedure, the listing shows the following:

```
In main routine, after calling subproc
**************************************

CPUBLIC     Pub    C  "public from sub"
CPUBLICPARM
            Pub    C  "ghi"
CLOCAL      Local  C  "abc"  testlistmemo
CLOCALPARM  Local  C  "abc"  testlistmemo
CPRIVATE    Priv   C  "def"  testlistmemo
CPRIVATEPARM
            Priv   C  "def"  testlistmemo
CPRIVATEINMAIN
            Priv   C  "def"  testlistmemo
```

The only item worthy of note is that the public variable, cPublic, retains the value assigned in the procedure, demonstrating that public really means public.

What does all this mean for you in terms of understanding your error log? Most importantly, it's a reminder than even declaring a variable

private doesn't guarantee that it will be available in called routines. Since depending on the existence of variables outside a function or procedure is bad practice, that's a good reminder.

As a general practice, use only local variables, passing parameters or using object properties when needed to move values between routines. Never use public variables. If you really need something (such an application object) to be visible throughout your application, use a private variable created in your main program, and be careful not to declare a variable of the same name.

The demonstration program, TestListMemo.PRG, is included on this month's PRD.

–Tamar