The source code for this article is available on Data Based Advisor's Forum on CompuServe (GO DBA). Call (800) 848-8199, Representative 201, for your free introductory Membership.

Flex Reports, Part 2

These tricks show how to get FoxPro user-defined functions to snake information across columns in your reports... ■ *By Lisa C. Slater*

Last time, we discussed variable-width reports using the IIF() function and arrays to handle situations in which you don't know how many columns your report will contain. In your next Report Writer challenge, the FRX file will know how many columns it has—now all we have to do is gather information from different detail records in the SELECTed database to fill them.

Why multiple records per line?

Suppose you have a list of account codes and their descriptions as a reference list. Together, these two fields might be only 30 columns wide. Nobody likes a reference sheet that runs for 10 mostly-blank pages.

Or perhaps you want to show all individual sales for the day, with a total at the end, in a point-of-sale system. You could do it with "paper-tape" printers, with paper only a few inches wide; otherwise you'll end up with mostly blank pages again.

Maybe you just want a list of names and addresses for your customers, and you'd like them more than one across on a page, to conserve paper. You could use the Label Generator, but then you'd have to hard code your page numbers and other headers and footers.

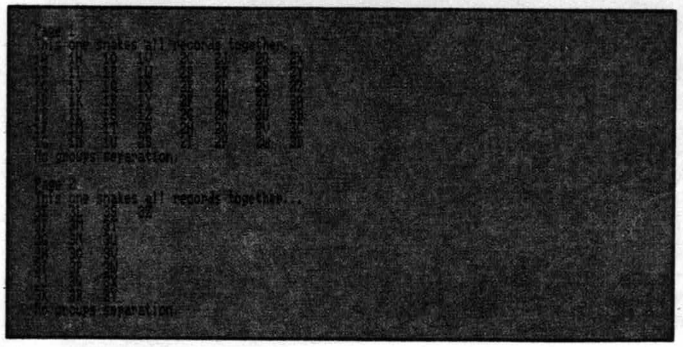
In each of these cases, the best solution is to put multiple detail records on each line. The key to doing it with the Report Writer is a few user-defined functions (UDFs).

Snaked and multiple columns

I use three different approaches to solve three different layout problems, shown in Figs. 1 to 3. In the first layout, records are consecutive across the page; in the second, all records in the report "snake" down the page and then up to the next column; and in the third, records "snake" a single group at a time. Listing 1 shows you the UDFs that make these layouts possible.

Figure 1—Multiple records on a line

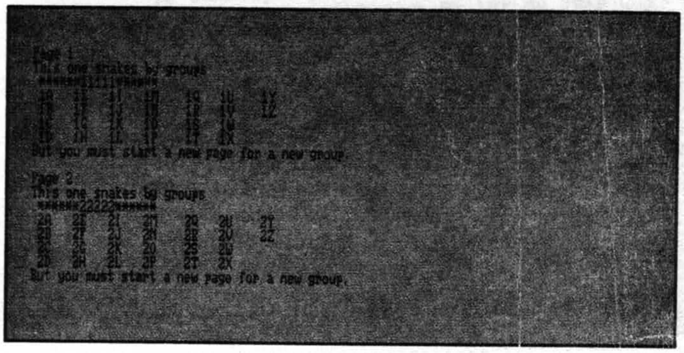
Figure 2—Snaking columns with no groups



```
Page 1
This one snakes all records together.
No groups seperation.

Page 2
This one snakes all records together...
No groups seperation.
```

Figure 3—Snaking columns grouped



```
Page 1
This one snakes by groups
But you must start a new page for a new group.

Page 2
This one snakes by groups
 ******22222******
 2A   2E   2I   2M   2Q   2U   2Y
 2B   2F   2J   2N   2R   2V   2Z
 2C   2G   2K   2O   2S   2W
 2D   2H   2L   2P   2T   2X
But you must start a new page for a new group.
```

Listing 1—The code to snake records across the page

```
*:*****************************************************
*:    Program: SNAKEIT.PRG      :: a procedure file
*:    Purpose: Multiple records per line in reports
*:    Author:  Lisa C. Slater
*:
*:    Fncts: DO_SKIP():: chain items across page
*:         : SNAKE2() :: snake items down by page
*:         : SNAKE3() :: snake items by group by page
*:         : D_LINES():: figure detail lines per page
*:*****************************************************

*:*****************************************************
*:  DO_SKIP() :: chain items across
*:*****************************************************

** Before calling the report:
** Name your group expression:
*  whichgroup = "test-group"
** If you're not grouping, assign a dummy variable
** that will never change value during report run:
*  mdummy = "X"
*  whichgroup = "mdummy"

FUNCTION do_skip
mreturn = .T.
oldvalue = &whichgroup
SKIP
* if we've changed group, need to let Report Writer
* do its own break
* need check for EOF() just in case we're not
* grouping so first condition will never be true
IF &whichgroup <> oldvalue .OR. EOF()
   SKIP -1
   mreturn = .F.
ENDIF
RETURN mreturn

*:*****************************************************
*: SNAKE2() :: snake all items down
*:*****************************************************

** Before calling report:
*  no_rows = d_lines("test2.frx")
** see D_LINES() for additional comments
*  rowcount = 1  && must be 1
*  whatprint = "test->group+test->thing" && print
                                          && expr
```

Continued

```
FUNCTION snake2
PARAMETERS no_col, no_cols
mreturn = ""
IF ! EOF()
   whichrec = RECNO()
   SKIP no_rows*(no_col-1)
   IF ! EOF()
      mreturn = &whatprint
   ENDIF
   IF no_cols = no_col
      rowcount = rowcount + 1
   ENDIF
   IF rowcount = no_rows + 1
      rowcount = 1
   ENDIF
   IF ! (rowcount = 1 .AND. no_cols = no_col)
      GO whichrec
   ENDIF
ENDIF
RETURN mreturn

*:******************************************************
*:  SNAKE3() :: snake items by group
*:******************************************************

** Report form *must* start new page on group expr
** Before calling report:
*  no_rows = d_lines("reportname.frx")
** see D_LINES() for additional comments
*  rowcount = 1  && must be 1
*  whatprint = ;
*  "test->thing+test->group"   && print expr
*  whichgroup = "test->group"  && group expr

FUNCTION snake3
PARAMETERS no_col, no_cols
mreturn = ""
IF ! EOF()
   whichrec = RECNO()
   thisgroup = &whichgroup
   SKIP no_rows*(no_col-1)
   IF &whichgroup == thisgroup .AND. ! EOF()
      mreturn = &whatprint
      IF no_cols = no_col && end of the line
         rowcount = rowcount + 1
         IF rowcount = no_rows + 1
            rowcount = 1
         ENDIF
      ENDIF
    ELSE
       IF no_cols = no_col && end of the line
          rowcount = rowcount + 1
          IF rowcount = no_rows + 1
             rowcount = 1
             IF &whichgroup <> thisgroup
                * need to let Report Writer do skip
                * as a break on the group
                DO WHILE &whichgroup <> thisgroup
                   SKIP -1
                ENDDO
             ENDIF
          ENDIF
       ENDIF
   ENDIF
   IF ! (rowcount = 1 .AND. no_cols = no_col)
      GO whichrec
      * otherwise let Report Writer do skip
   ENDIF
ENDIF
RETURN mreturn

*:******************************************************
*:  D_LINES() :: figure detail lines per page
*:******************************************************

* Doesn't work with blank header or footer lines!
```

only Total that information. Instead of using the built-in Totaling options, however, you can easily write a UDF to return the appropriate result from all records in the group, or build the calculation directly into the UDFs that do the SKIPs.

Stretching the band

I often use a string variable to indicate what should actually print, although you can have the UDFs RETURN ".T." or ".F." and have the report form contain the print expressions, as above. But leaving the print expression outside the report makes it easier to use these reports for many different purposes, with some minor editing. I merely need to determine the expression width for a new print expression and delete or add a few columns accordingly.

What you print in each expression can be more than one line with this technique. Create a variable like this:

```
whatprint = [addr-name+";"+addr-phone+";"+addr-custid]
```

Using @; as the expression format tells the Report Writer to move to the next line when it finds the semi-colon in quotation marks. Make your expression length the maximum of the longest line in your multi-line expression, and use the option to have each expression "stretch vertically."

Skipping down instead of across

The next two layouts (snaked columns) are a little more complicated. In addition to group and print expressions, you need to initialize a row counter to 1 before calling your report. After checking for a current EOF(), each UDF contains the expression:

```
SKIP no_rows*(no_col-1)
```

to move from the left-most record on a line to the one that should print at the current page position. Then it does a variety of checks to see whether there's a record that should be printed there. It checks for the end of a line and the end of a page and changes the row counter accordingly. Then it moves the record pointer again, back to where it was on the left-most record, unless it's reached the end of a page. If it has, the Report Writer will do a SKIP for the first record on the next page.

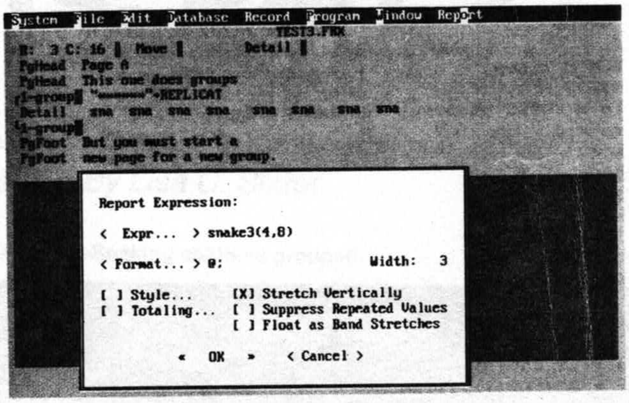
Since we use the whatprint variable to return the actual print expression, the report expressions are relatively simple. As shown in Fig. 4, each expression takes the form of the UDF with two parameters; the first parameter indicates the relative position of this column, the second indicates the total number of columns in the layout.

Stretching the edge of the envelope

Both snaking layouts use the UDF called D_lines(). As indicated in the comments, D_lines() figures out how many detail lines will be available on each report page, using FoxPro's low-level file functions to examine each line of an FRX file. It checks the first two positions of each line to see what object type the line describes. An object type of 1 indicates the report "header," with margins and page length noted. An object type of 18 indicates band information that, for D_lines()'s purposes, can be ignored. All other objects must be checked for their height, and the number of detail lines available adjusted.

These techniques have a few other limitations. SNAKE3() requires a page break on your group to calculate properly, and D_lines() doesn't account for any blank header or footer lines your report may contain, or headers or footers that stretch vertically.

Figure 4— The report expression UDF requires two parameters: one for the column position, one for the number of columns.



```
                         TEST3.FRX
R:  3 C: 16 | Move |          Detail |
PgHead  Page #
PgHead  This one does groups
1-group| "--------"+REPLICAT
Detail   sna  sna  sna  sna   sna  sna   sna  sna
1-group|
PgFoot  But you must start a
PgFoot  new page for a new group.

   Report Expression:

   <  Expr...   > snake3(4,8)

   < Format... > @;                     Width:   3

   [ ] Style...      [X] Stretch Vertically
   [ ] Totaling...   [ ] Suppress Repeated Values
                     [ ] Float as Band Stretches

            «   OK   »    < Cancel >
```

If you use the trick with semi-colons to have multiple lines in each expression, you'll need to get the number of detail lines as follows:

```
no_rows = D_lines("test2.frx")/;
                       (OCCURS(";",whatprint)+1)
```

Use the MOD() function on the result of this expression to make sure it comes out even; if not, you can write a procedure using the functions FCHSIZE() and FPUTS() to add a few page footer lines to the FRX until it does.

You can use the same trick even if your details are one line each, to allow more than one snaked "block" on a page. Again, you must make sure that you aren't left with a fractional value for no_rows.

Conclusions: What makes this easier?

We've used macro expressions, arrays, UDFs, the IIF() function, and @; formatting; we've moved the record pointer and used FoxPro's low-level file functions. I've yet to find a situation where I couldn't figure out a way to use these techniques, along with Report Writer's tools, to solve a reporting problem. What's the advantage to this over hard-coding, considering that by writing UDFs and complex report expressions we *are* hard-coding some of the report?

We can concentrate on report *design*; we can remain focused on how we want to present the information—and what we want it to communicate. I leave the mundane, yet painful, details—page numbers, grouping, most box drawing, and (most importantly) page coordinates for each object—to the Report Writer where they belong.

Some parts of creating complex reports are immediately rewarding and intrinsically interesting, others must be done routinely and have no particular charm. One of our challenges is to determine which part is which, maximize our time with the former, and perform the latter with dispatch.

To me, it's worth the money to find a gardener who trims all my trees correctly, with power tools, in half an hour so I can spend more time experimenting with exotic species of basil It's also worth my time to learn to use the FoxPro Report Writer, the professional "page-trimmer," that's always available. It can leave me free to enjoy—not abandon—cultivation of my application gardens.

Lisa C. Slater is a writer and computer consultant raising unusual children, herbs, and FoxPro code. You can reach her on CompuServe (72077,2417). ☐