March, 2004

Advisor Answers

Counting groups in a report

VFP 8/7/6

Q: I want to add colored bars to a report created in the Report designer (kind of similar to green bar paper) to make a report with a bunch of tiny numbers stand out a bit more.

I'm using the rectangle tool with a gray background. The rectangle has a Print When condition based on a variable called linecounter (check whether it's odd or even to determine when to print). This works great for a report where all the details are printed line by line. I get a nice bar effect on every other line.

The problem comes when I try to do this same thing on a summary/totals report. The totals are in the group (a single group in this report) and I've emptied the detail band, just keeping the group. The line counter is actually counting detail records and ends up giving weird results.

I've tried using PROW() to figure out what row the printer is actually on, but it just returns 0. Is there a way that FoxPro keeps track of the current line number on the report that I could use to check even/odd?

A: Your question touches on a variety of topics, so I'll take them one at a time to arrive at a complete solution. Let me start with your final question about tracking a line number. As you discovered, the PROW() function is meaningless in a VFP report. (Amazingly, it does still work with output generated using @SAY.)

Next, there's no way to find the position of the print head in a VFP report. Given the complexity of the Windows' printing system, this isn't really surprising.

Fortunately, there's another approach you can use to solve your problem. What you're really trying to do here is count groups rather than details lines. You can do so using a pair of report variables.

To make the technique easier to explain, let's consider a concrete example using the TasTrade database. Consider a report showing sales by product category. You might use this query (CategorySales.PRG on this month's Professional Resource CD) to collect the data. It produces one record for each sale of each product, ordered by category and product:

```
SELECT Order_Date, Order_Line_Items.Product_id, ;
       Category.Category_id, ;
       Order_Line_Items.Unit_Price*Quantity AS nSales, ;
       Category_Name ;
  FROM Orders ;
    JOIN Order_Line_Items ;
      ON Orders.Order_id = Order_Line_Items.Order_id ;
    JOIN Products ;
      ON Products.Product_id=Order_Line_Items.Product_id;
    JOIN Category ;
      ON Products.Category_ID = Category.Category_ID ;
  ORDER BY 3, 2 ; && Category_ID, Product_ID
  INTO CURSOR Result
```

To report on this data by category, create a report (GreenBar.FRX on this month's PRD) and add a data group with a grouping expression of Category_ID. Put no fields into the detail band (in fact, shrink the detail band to take up no space). In the Group Footer band, add a field with the expression set to nSales, Calculate set to sum, and Reset set to Category_ID.

Add a rectangle to the Group Footer band, enlarge it to fill the band, set its background color to gray (or whatever color you want), and then send the rectangle to the back, so that the total sales shows on top of it.

If you run the query, then run the report at this point, you have a gray background behind all the data. The next step is to provide a way to alternate the background.

The technique takes advantage of the fact that report variables are evaluated in the order you list them in the Report Variables dialog. To know when you start a new group, you need to define two variables in the right order.

Call the first variable lNewGroup. Set its Initial value to .T. (to indicate that you're starting with a new group). Set its Value to store to:

```
IIF(cLastGroup == Category_ID, .F., .T.)
```

The second variable is cLastGroup; it's set up to hold the value of the grouping variable on the previous pass. Set its Initial value to something that can't occur as the grouping variable (the empty string, in the example). Set its value to store to the grouping expression, Category_ID.

What happens when the report runs is that both variables are initialized before any output is generated. The Value to store for each variable is evaluated each time you reach a new record. However, lNewGroup is evaluated first, so when you reach the first record in a new category, cLastGroup still contains the id for the last category and the comparison fails, setting lNewGroup to .T.

To put the gray rectangles on every other group, the next step is to add another report variable. Call this one nLineCount, set its Initial value to 0, and set its Value to store to:

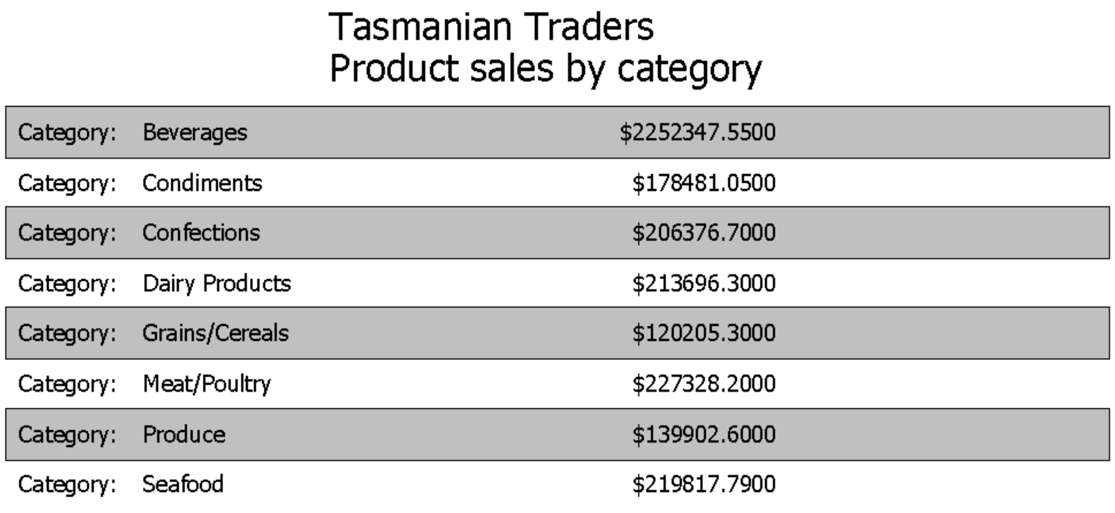
```
IIF(lNewGroup, 1, 0)
```

Set this variable to Calculate the Sum. As you move through the report, nLineCount goes up each time you reach a new group, which in this case is on each line of the report.

Set a Print When condition for the rectangle to:

```
MOD(nLineCount, 2) = 1
```

That includes the rectangle for all the odd-numbered groups and omits it for the even ones. Figure 1 shows the green bar report.



Tasmanian Traders
Product sales by category

Category:	Beverages	$2252347.5500
Category:	Condiments	$178481.0500
Category:	Confections	$206376.7000
Category:	Dairy Products	$213696.3000
Category:	Grains/Cereals	$120205.3000
Category:	Meat/Poultry	$227328.2000
Category:	Produce	$139902.6000
Category:	Seafood	$219817.7900

Figure 1. Simulating green bar paper—A shaded rectangle, along with a few report variables, lets you create reports with alternating shading.

The technique for determining whether you're in a new group or not has many other uses. For example, you can use it to determine whether to put "Continued" in a Group Header band.

Counting the number of groups (as with nLineCount here) is useful, especially in reports with multiple levels of grouping. You can count the number of inner groups to include in a group footer of an outer group. For example, using the same data as in this report, you could include an inner grouping on Product_ID and count the number of distinct sales of each product to include in the Product footer.

Finally, it's worth noting that there's actually a much simpler solution for my example of reporting on sales by category. Instead of collecting data by individual item sold, a query could consolidate the data by category, like this:

```
SELECT Order_Line_Items.Product_id, ;
    Category.Category_id, ;
    SUM(Order_Line_Items.Unit_Price*Quantity) AS nSales,;
    Category_Name ;
  FROM Orders ;
    JOIN Order_Line_items ;
      ON Orders.Order_id = Order_Line_Items.Order_id ;
    JOIN Products ;
      ON Products.Product_id=Order_Line_Items.Product_id;
    JOIN Category ;
      ON Products.Category_ID = Category.Category_ID ;
  ORDER BY 2, 1 ;
  GROUP BY 2, 4, 1 ;
  INTO CURSOR Result
```

Then, the report could put the category data into the detail band and avoid the whole issue of grouping. However, there are times when performing all the computations prior to running a report is difficult. In those situations, report variables often let you get the results you need.

–Tamar