

# Handling crosstabs and other wide data in VFP reports

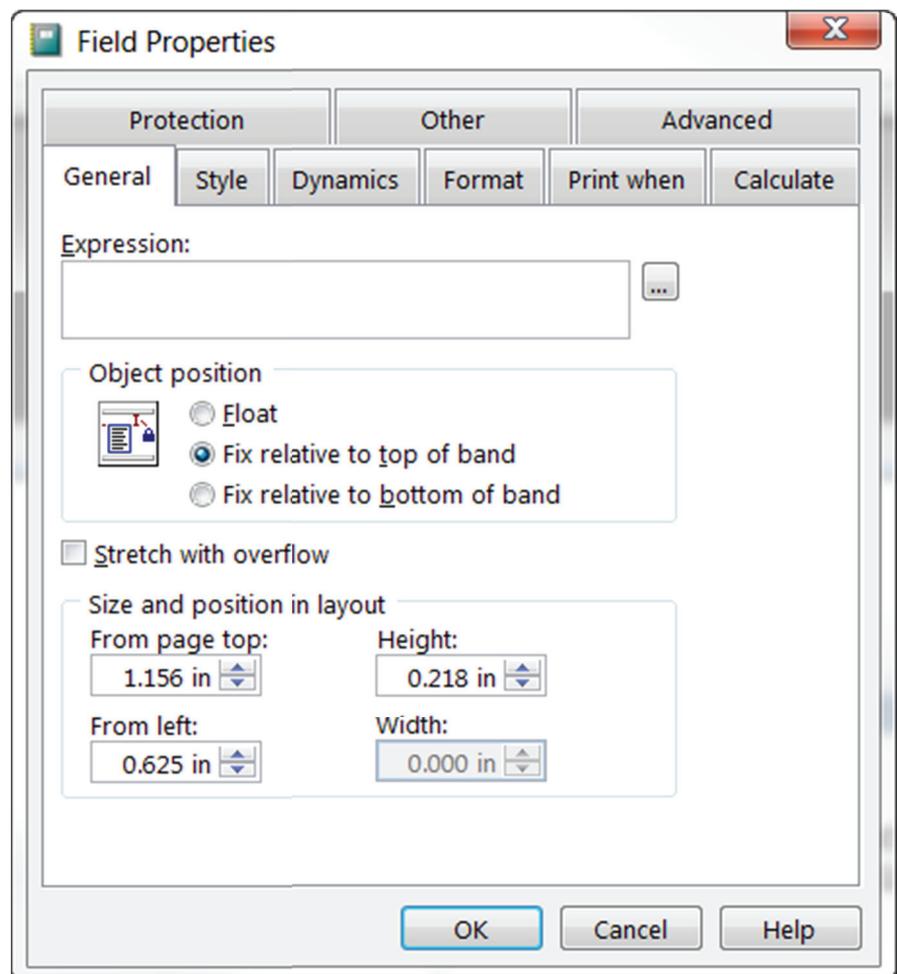
*When the data you want to report on has many columns, you have a few options. One takes advantage of VFP's flexibility.*

**Tamar E. Granor, Ph.D.**

In the May, July, and September, 2016 issues, I looked at VFP's crosstabs and SQL Server's PIVOT functionality. Both allow you to turn rows into columns, usually to aggregate data. The result may have many columns (and in some cases, you may not know their names), which makes traditional reporting difficult. In this article, we'll start to look at options for reporting on crosstab and pivot data, and in fact, any kind of wide data, from VFP.

Traditional reporting in a VFP application involves creating reports using the Report Designer. For most application data, creating such reports is straightforward. You simply create a column in the report for each field of interest. But crosstab/pivot data makes that difficult. At the time the report is designed, we often won't know either how many data columns there are (since they're based on the data we find) or the names of those columns. In this article, I show a technique that I first learned roughly 20 years ago in a FoxTalk article by Nancy Jacobsen. Improvements in the Report Designer since then make the technique easier to use today.

Beyond VFP reports, customers today expect more. They may want to have data exported to Excel, or see graphs and charts. In future articles, I'll look at how to send your crosstab and pivot data to those.



**Figure 1.** The Field Properties dialog lets you specify what a Field control in a report shows. Most often, it's just a field of a table or cursor.

## The Problem with Reports

To create a report in VFP's Report Designer, you drop controls onto the report and position them as desired. (There's lots more you can do, but that's the core activity.) To include fields, you use the Field control, which lets you specify an expres-

sion using the dialog shown in **Figure 1**. For most reports, the expression for most Field controls is simply a field of a table or cursor. Sometimes, you add formatting with TRANSFORM() or combine a table field with some fixed text or combine multiple fields into a single report field.

The key point is that the expression includes the name of the field. When reporting on a crosstab, you may not know the name of each field in the cursor at the time you design the report.

Even trickier, because the list of fields in the crosstab result depends on the values in the original data, you may not know how many fields are in the crosstab result. In addition, it's possible to have more fields than fit on a single page.

For example, the query in **Listing 1** (repeated from my May, 2016 article and included as SalesPersonAnnualSumAvgCnt.PRG in this month's downloads) computes the total sales, average sale and number of sales for each employee for each year. The number of columns in the result depends on the number of years for which there is data. Columns have names like N\_1997 (for total sales in 1997). (To run this code and the other examples in this article, you need to make sure FastXTab.PRG is in the path or add the path to each reference to the program in the code. You can download FastXTab, discussed in my May, 2016 article from <http://praisachion.blogspot.com/2015/02/fastxtab-version-16.html>.) **Figure 2** shows partial results.

**Listing 1.** This code creates a crosstab of sales by employee by year. For each employee, it has total sales, average sale and number of sales.

```
SELECT EmployeeID, Orders.OrderID, OrderDate,;
      SUM(Quantity*UnitPrice) AS OrderTotal ;
FROM Orders ;
  JOIN OrderDetails ;
  ON Orders.OrderID = ;
      OrderDetails.OrderID ;
GROUP BY 1, 2, 3 ;
INTO CURSOR csrOrderTotals

LOCAL oXTab AS FastXTab OF "fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", "fastxtab.prg")
WITH oXTab AS FastXTab OF "fastxtab.prg"
  .cROWFIELD = 'EmployeeID'
```

```
.cColFIELD = 'YEAR(OrderDate) '
.nMULTIDATAFIELD = 3
.acDatafield[1] = 'OrderTotal'
.anFunctiontype[1] = 1
.acDataField[2] = 'OrderTotal'
.anFunctiontype[2] = 3
.anDatafield[3] = 'OrderID'
.anFunctiontype[3] = 2
.couTFIELD = "csrXtab"
.lCursorOnly = .T.
.lCLOSETABLE = .F.
.RunXtab()
ENDWITH
```

## FIELD() and EVAL() to the rescue

The solution to not knowing the field names is to not put them into the report. Instead, use the FIELD() function. As the syntax shown in **Listing 2** indicates, FIELD() accepts two parameters. The first is the field number; it's required. The second parameter is optional and specifies the table you're interested in.

**Listing 2.** The FIELD() function lets you refer to fields without knowing their names.

```
cFieldName = FIELD( nFieldName
                  [, cAlias | nWorkarea ])
```

So rather than referring to N\_1997 in the report, we can refer to FIELD(6). Using FIELD() solves the problem of not knowing the actual field names.

FIELD() gives us the name of the field. To determine its contents, use the EVAL() function. EVAL() is short for EVALUATE() and that's what it does. The function evaluates the expression you pass to it and returns the result. (I wrote at length about EVAL() and related language in the May, 2009 issue.)

So to get the value of N\_1997 for the current record, use EVAL(FIELD(6)).

## How many fields?

The other problems are that we don't know how many columns we need in a report and we don't know whether they'll fit on a single page. We solve these two together by creating a report with as many fields as fit and then running it as many times as necessary to show all fields. Doing so requires a combination of a report and code to control it.

| Employeeid | N_1996     | N_1996_2 | N_1996_3 | N_1997      |
|------------|------------|----------|----------|-------------|
| 1          | 38789.0000 | 1491.885 | 26       | 97533.5800  |
| 2          | 22834.7000 | 1427.169 | 16       | 74958.6000  |
| 3          | 19231.8000 | 1068.433 | 18       | 111788.6100 |
| 4          | 53114.8000 | 1713.381 | 31       | 139477.7000 |
| 5          | 21965.2000 | 1996.836 | 11       | 32595.0500  |
| 6          | 17731.1000 | 1182.073 | 15       | 45992.0000  |
| 7          | 18104.8000 | 1645.891 | 11       | 66689.1400  |
| 8          | 23161.4000 | 1219.021 | 19       | 59776.5200  |
| 9          | 11365.7000 | 2273.140 | 5        | 29577.5500  |

**Figure 2.** The code in **Listing 1** produces a cursor with three columns for each year.

We'll return to the example in Listing 1 later, but first let's consider a simpler example. The code in Listing 3 computes sales for each employee for a given year (1997) by month. There's one column in the result for each month of the year. In this case, we know how many columns there are and their names, but they still don't fit onto a single page. Figure 3 shows partial results; in keeping with the theme of this article, they're cut off on the right.

**Listing 3.** This crosstab computes sales by employee by month for a single year.

```
SELECT EmployeeID, OrderDate, ;
       SUM(Quantity*UnitPrice) AS OrderTotal ;
FROM Orders ;
  JOIN OrderDetails ;
  ON Orders.OrderID = ;
  OrderDetails.OrderID ;
GROUP BY 1, 2 ;
INTO CURSOR csrMonthlyTotals

LOCAL oXTab AS FastXTab OF "fastxtab.prg"

oXTab = NEWOBJECT("fastxtab", "fastxtab.prg")
WITH oXTab AS FastXTab OF fastxtab.prg"
  .cRowField = 'EmployeeID'
  .cColField = 'MONTH(OrderDate)'
  .cDataField = 'OrderTotal'
  .cOutFile = "csrXtab"
  .cCondition = 'year(OrderDate) = 1997'
  .lCursorOnly = .T.
  .lCloseTable = .T.
  .RunXtab()
ENDWITH
```

| Cname            | Employeeid | N_1        | N_2        |
|------------------|------------|------------|------------|
| Steven Buchanan  | 5          | 0.0000     | 0.0000     |
| Laura Callahan   | 8          | 6701.1000  | 7764.4000  |
| Nancy Davolio    | 1          | 7331.6000  | 2504.6000  |
| Anne Dodsworth   | 9          | 1208.5000  | 0.0000     |
| Andrew Fuller    | 2          | 3150.2000  | 1584.0000  |
| Robert King      | 7          | 13703.4000 | 3891.0000  |
| Janet Leverling  | 3          | 7477.9000  | 10581.3000 |
| Margaret Peacock | 4          | 25620.1000 | 13530.3000 |
| Michael Suyama   | 6          | 1500.0000  | 1351.6000  |

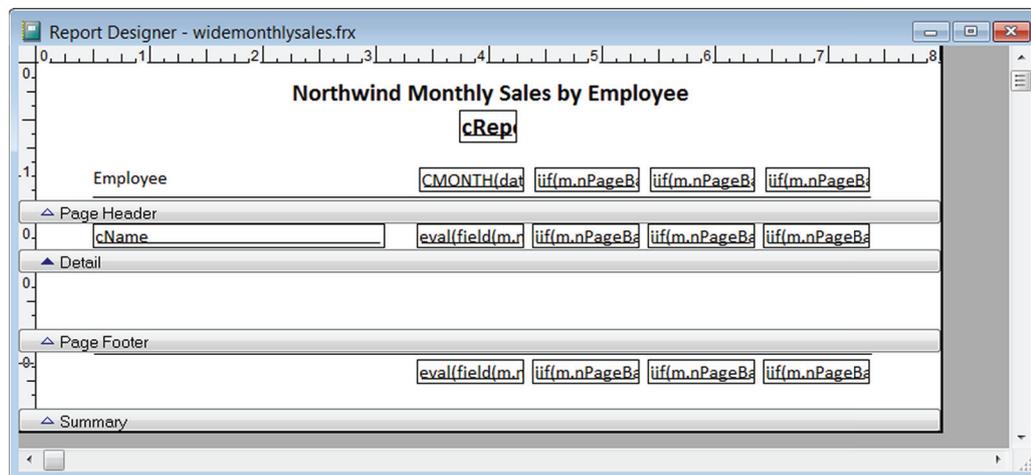
**Figure 3.** This cursor contains sales for each employee for each month of a single year.

```
* Add employee name
SELECT PADR(ALLTRIM(FirstName) + ;
          (' ' + LastName), 30) AS cName, ;
       csrXtab.* ;
FROM csrXtab ;
  JOIN Employees ;
  ON csrXtab.EmployeeID = ;
  Employees.EmployeeID ;
ORDER BY LastName, FirstName ;
INTO CURSOR csrReport
```

We can refer to the data fields in this cursor as FIELD(3), FIELD(4), and so on, but how can we get them all into a report?

First, we need to figure out how many data columns we can fit on a page. I decided to keep the paper in portrait mode (you can fit more columns in landscape, of course) and a little experimentation showed me that along with the employee's name, four data columns would fit. Figure 4 shows the report; we'll dig into its contents later in this article.

Once we know how many columns fit, we can figure out how many pages we'll need. (I'm using the word "page" here to indicate pages across. It's entirely possible that there could be enough records in the cursor to print multiple pages of length, as well.) In the code to drive the report, we can set a variable, nFieldsPerPage, to that value (4, in this case). We also need to know where in the cursor the data fields start. In this case, they start in column 3, because column 1 is the employee name



**Figure 4.** We can fit the employee's name and four months' data on one page in portrait mode.

and column 2 is the employee number. Store that number in a variable as well; I call it nFirstDataCol. With that information, we can divide the total number of data fields by fields per page, to determine how the number of pages. Listing 4 shows this part of the code.

**Listing 4.** To make the report work, we need to ask some questions about the data: how many fields there are total, where the data fields start, and how many fields fit on a page. Using those, we can determine the number of pages we need.

```
nFieldCount = FCOUNT("csrReport")
nFirstDataCol = 3
nFieldsPerPage = 4
nPages = CEILING((m.nFieldCount - ;
  m.nFirstDataCol)/m.nFieldsPerPage)
```

Obviously, though, we only want to create a single report, not a separate report for each page. So we need to specify the fields and headings in the report in a way that works for each page. Using FIELD() is clearly part of the answer, but we also need to calculate the parameter we pass to FIELD(), so that for each page of the report, we get the right subset of fields. **Listing 5** shows the code that actually runs the report. The key is the calculation of nPageBase, a variable that determines the position of the first data column on each page. For our example, on the first page, nPageBase = 3, the first data column. On the second page, nPageBase = 7, and so on. We can use nPageBase in the report to figure out which fields appear on this page.

**Listing 5.** We run the report in a loop, once for each page.

```
FOR m.nPage = 1 TO m.nPages
  nPageBase = (m.nPage-1) * m.nFieldsPerPage ;
    + m.nFirstDataCol

  REPORT FORM WideMonthlySales PREVIEW
ENDFOR
```

## Making the report generic

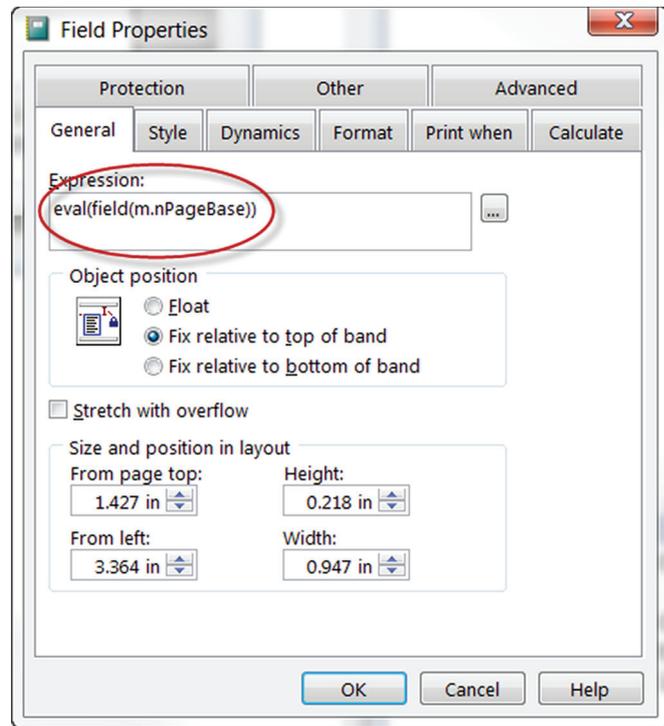
The final piece is to design the report to use the pieces we've collected. The column headings, the fields themselves and any totals need to be generic.

We can assume that we only print a page if we have at least one data column to show there, but any other data column may or may not appear on the last page. In the sales by month example we're working on, there are 12 data columns (one per month) and 4 data columns per page, so in fact, this isn't an issue. But we'll look at how to handle it anyway.

In the first data column, to show the value, we use EVAL(FIELD(m.nPageBase)), as in **Figure 5**.

For the other data columns, we need to do two things. First, we need to count upward from nPageBase, so the second data column will show data from FIELD(m.nPageBase + 1), the third from FIELD(m.nPageBase + 2) and so on.

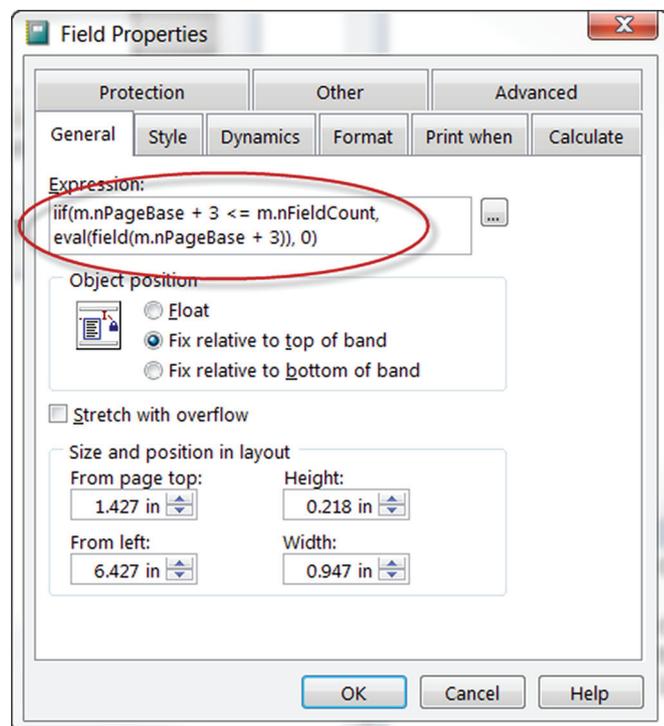
Second, we need to consider the possibility that we've reached the end of the data columns. To handle that case, we wrap the expression in IIF(), as in **Listing 6**, which shows the expression for the fourth data column. **Figure 6** shows the expression in the Field Properties dialog.



**Figure 5.** The expression for the value of the first data field on the page is fairly simple because we know that it will always be present.

**Listing 6.** After the first data column on the page, we have to take into account the possibility that there is no such column.

```
iif(m.nPageBase + 3 <= m.nFieldCount,
  eval(field(m.nPageBase + 3)), 0)
```



**Figure 6.** For data columns after the first, we use IIF() so that we don't try to evaluate a field that doesn't exist.

As the expression indicates, though, we're still returning 0 when there's no field. We want to make sure nothing appears in a column for which there's no corresponding field. To do that, we use the Report Designer's Print When capability, specifying the same condition we used in IIF(). **Figure 7** shows the dialog.

For column totals, you apply the same techniques, using EVAL(FIELD()), adding IIF() for data columns after the first, and setting the total field to print only when the column exists.

Column headings use similar techniques, but need a little creativity because the field names in a crosstab may not be terribly informative. In this example, they look like N\_1, N\_2, etc. We know that the number at the end is the month number we're interested in, so we can use that to build an informative heading.

As before, we know we'll have data in the first column on the page. For the first data column, we use the expression in **Listing 7**. Working from the inside out, we grab the field name and extract everything after the underscore. We convert that from character to numeric, and then use it as the month parameter to DATE(). Finally, we call CMONTH() to get the name of the month.

**Listing 7.** To get a meaningful column heading, we extract the numeric part of the field name, build a date using that number for the month and then get the month name.

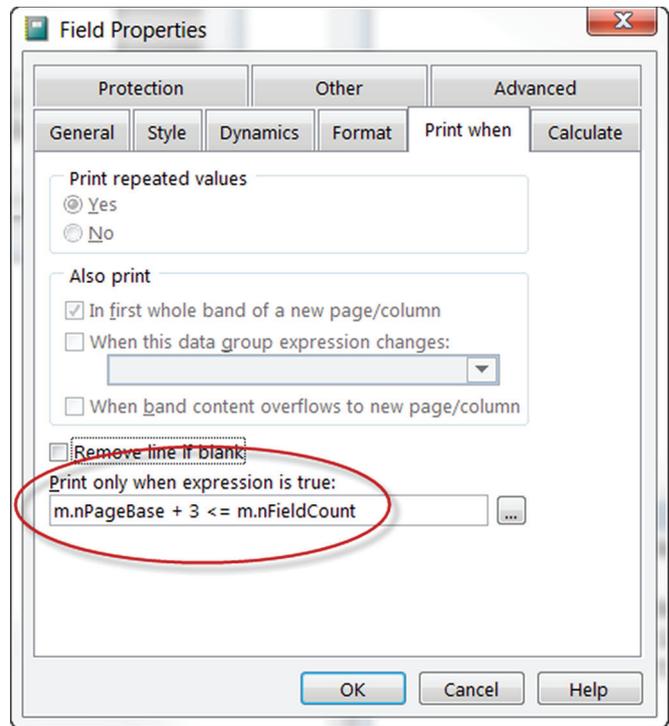
```
CMONTH(date(m.nReportYear,
val(strextract(field(
    m.nPageBase), "_"), 1))
```

For subsequent columns, we need to wrap the heading in the same IIF() condition we used for the data values and totals, and specify Print When to make the column appear only when it actually exists.

For this example, that's everything. **Figure 8** shows the first page of the report; **Figure 9** shows the last (third) page. This month's downloads include ReportSalesPersonMonthly.prg, which puts the whole process together, including running the report, and WideSalesAvgCount.frx, the report itself.

## Handling more complicated data

Sometimes, as in Listing 1, the data created by the crosstab is more complex. That code creates a cursor with three data columns for each year. In the report, we want all three of those columns on the same page. So we need



**Figure 7.** To prevent anything from showing up for a non-existent column, we use the Print When tab of the Field Properties dialog.

| Employee         | January      | February     | March        | April        |
|------------------|--------------|--------------|--------------|--------------|
| Steven Buchanan  | \$ 0.00      | \$ 0.00      | \$ 2,634.40  | \$ 0.00      |
| Laura Callahan   | \$ 6,701.10  | \$ 7,764.40  | \$ 4,806.10  | \$ 800.50    |
| Nancy Davolio    | \$ 7,331.60  | \$ 2,504.60  | \$ 5,493.90  | \$ 240.00    |
| Anne Dodsworth   | \$ 1,208.50  | \$ 0.00      | \$ 1,770.80  | \$ 611.00    |
| Andrew Fuller    | \$ 3,150.20  | \$ 1,584.00  | \$ 2,905.10  | \$ 14,019.30 |
| Robert King      | \$ 13,703.40 | \$ 3,891.00  | \$ 3,867.20  | \$ 5,707.35  |
| Janet Leverling  | \$ 7,477.90  | \$ 10,581.30 | \$ 11,599.40 | \$ 10,297.35 |
| Margaret Peacock | \$ 25,620.10 | \$ 13,530.30 | \$ 5,644.80  | \$ 14,333.15 |
| Michael Suyama   | \$ 1,500.00  | \$ 1,351.60  | \$ 1,258.20  | \$ 9,690.74  |
|                  | \$ 66,692.80 | \$ 41,207.20 | \$ 39,979.90 | \$ 55,699.39 |

**Figure 8.** When we run the loop, we see each (width) page of the report one at a time.

| Employee         | September    | October      | November     | December     |
|------------------|--------------|--------------|--------------|--------------|
| Steven Buchanan  | \$ 2,091.70  | \$ 8,365.20  | \$ 509.75    | \$ 629.50    |
| Laura Callahan   | \$ 2,891.40  | \$ 11,598.17 | \$ 4,337.50  | \$ 4,728.10  |
| Nancy Davolio    | \$ 8,461.50  | \$ 12,920.15 | \$ 4,106.70  | \$ 16,077.25 |
| Anne Dodsworth   | \$ 10,412.40 | \$ 378.00    | \$ 7,398.05  | \$ 1,941.50  |
| Andrew Fuller    | \$ 9,622.25  | \$ 10,164.80 | \$ 3,652.50  | \$ 7,834.75  |
| Robert King      | \$ 13,839.29 | \$ 642.00    | \$ 1,990.00  | \$ 928.00    |
| Janet Leverling  | \$ 3,595.50  | \$ 8,572.75  | \$ 9,668.06  | \$ 18,494.00 |
| Margaret Peacock | \$ 8,147.63  | \$ 10,996.53 | \$ 7,684.75  | \$ 18,843.25 |
| Michael Suyama   | \$ 671.35    | \$ 6,690.90  | \$ 6,566.05  | \$ 7,999.91  |
|                  | \$ 59,733.02 | \$ 70,328.50 | \$ 45,913.36 | \$ 77,476.26 |

**Figure 9.** In this example, the final page has the same number of columns as the others, but that's not required.

to make sure that the number of data columns per page is a multiple of 3. I chose to put 6 data columns, two years' worth on a page. Figure 10 shows the report layout.

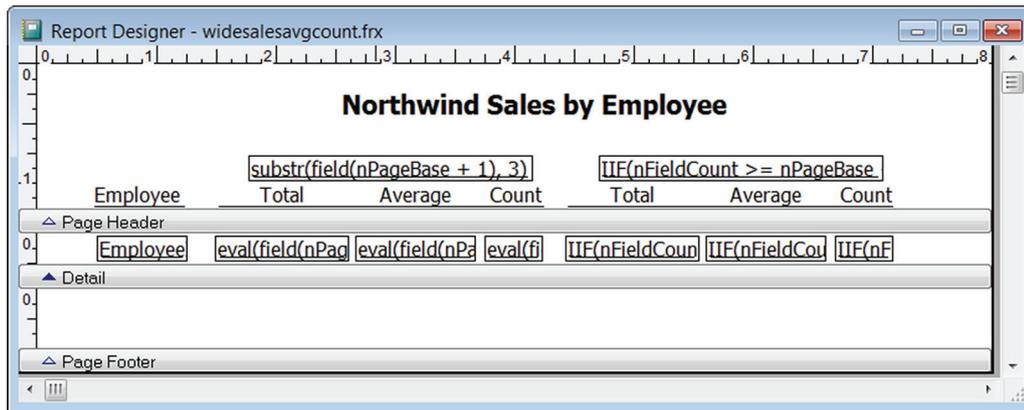


Figure 10. There's room for two years' data on one page in this report.

Ideally, we should group the column headings as well, and show the year above the three columns for that year. In this case, the field names are in the form N\_1996, etc., so it's easy to parse the year portion out and build an expression, as in Listing 8.

**Listing 8.** We want one column heading across all three columns for a year. This expression parses the year from the name of the first of the three fields for that year.

```
substr(field(nPageBase + 1), 3) + ' sales'
```

As before, after the first data column on a page, we need to make sure there's actually data. In this case, because we know the data columns come in groups of 3, we don't have to check until we get to

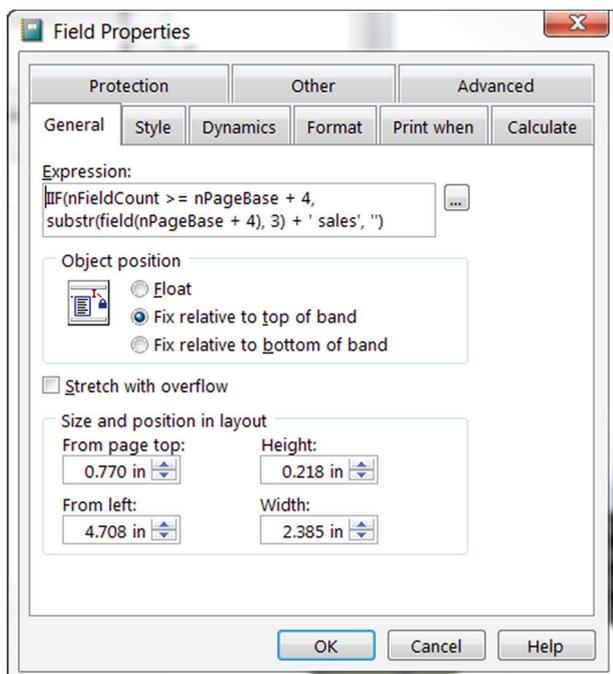


Figure 11. This heading for the second year's data on the page uses IIF() to make sure there is actual data to report.

column 4, the beginning of the second year's data on the page. Figure 11 shows the Field Properties dialog for the heading over the 4<sup>th</sup>, 5<sup>th</sup> and 6<sup>th</sup> data columns on the page.

This report also addresses an issue I ignored in the previous example, handling the lines under headings (and, if you have them between rows or before totals). You really only want lines where there's data. Here, each year's data has a separate line under its headings. The line for the second group on the

page has a Print When condition to ensure that it appears only when there's actually data. Figure 12 shows the Print When tab of the Line Properties dialog with the expression.

The data fields themselves use the same EVAL(FIELD(n)) approach as in the previous example. The fields for the second year on the page are wrapped with IIF() and have a Print When condition.

The code to drive the report is pretty much the same as in the previous example, figuring out how many fields there are and how many pages, then looping to run the report repeatedly. The complete

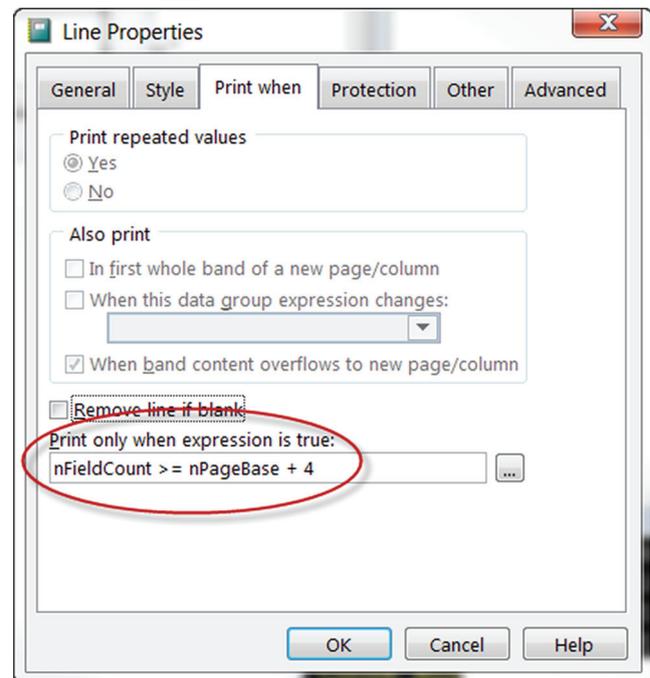


Figure 12. This Print When condition ensures that the underline for the second year's headings shows up only if there is a second year on the page.

program to drive this report is included in this month's downloads as reportsalespersonannualsumavgcnt.prg.

Figure 13 shows the first page of the report, while Figure 14 shows the final (second) page. Note that the right-hand side in Figure 14 is entirely blank, with no headings, no values, and no lines.

| Employee | 1996 sales   |             |       | 1997 sales   |             |       |
|----------|--------------|-------------|-------|--------------|-------------|-------|
|          | Total        | Average     | Count | Total        | Average     | Count |
| 1        | \$ 38,789.00 | \$ 1,491.88 | 26    | \$ 97,533.58 | \$ 1,773.33 | 55    |
| 2        | \$ 22,834.70 | \$ 1,427.16 | 16    | \$ 74,958.60 | \$ 1,828.25 | 41    |
| 3        | \$ 19,231.80 | \$ 1,068.43 | 18    | \$111,788.61 | \$ 1,574.48 | 71    |
| 4        | \$ 53,114.80 | \$ 1,713.38 | 31    | \$139,477.70 | \$ 1,721.94 | 81    |
| 5        | \$ 21,965.20 | \$ 1,996.83 | 11    | \$ 32,595.05 | \$ 1,810.83 | 18    |
| 6        | \$ 17,731.10 | \$ 1,182.07 | 15    | \$ 45,992.00 | \$ 1,393.69 | 33    |
| 7        | \$ 18,104.80 | \$ 1,645.89 | 11    | \$ 66,689.14 | \$ 1,852.47 | 36    |
| 8        | \$ 23,161.40 | \$ 1,219.02 | 19    | \$ 59,776.52 | \$ 1,106.97 | 54    |
| 9        | \$ 11,365.70 | \$ 2,273.14 | 5     | \$ 29,577.55 | \$ 1,556.71 | 19    |

Figure 13. The first page of the report includes data for two year.

| Employee | 1998 sales   |             |       |
|----------|--------------|-------------|-------|
|          | Total        | Average     | Count |
| 1        | \$ 65,821.13 | \$ 1,567.17 | 42    |
| 2        | \$ 79,955.96 | \$ 2,050.15 | 39    |
| 3        | \$ 82,030.89 | \$ 2,158.70 | 38    |
| 4        | \$ 57,594.95 | \$ 1,308.97 | 44    |
| 5        | \$ 21,007.50 | \$ 1,615.96 | 13    |
| 6        | \$ 14,475.00 | \$ 761.84   | 19    |
| 7        | \$ 56,502.05 | \$ 2,260.08 | 25    |
| 8        | \$ 50,363.11 | \$ 1,624.61 | 31    |
| 9        | \$ 42,020.75 | \$ 2,211.61 | 19    |

Figure 14. The last page of the report has data for only one year.

## Final thoughts

The technique in this article can be used for pretty much any data, as long as it's somewhat regular, that is, as long as there's a repeating set of one or more fields that run from some starting point in the table to the end.

Of course, a VFP report isn't the only way to show data. In future articles, I'll look at how to report on crosstab and pivot data in Excel and using graphs.

## Author Profile

*Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro and Taming Visual FoxPro's SQL. Her latest collaboration is VFPX: Open Source Treasure for the VFP Developer, available at [www.foxrockx.com](http://www.foxrockx.com). Her other books are available from Hentzenwerke Publishing ([www.hentzenwerke.com](http://www.hentzenwerke.com)). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at [tamar@thegranors.com](mailto:tamar@thegranors.com) or through [www.tomorrowssolutionsllc.com](http://www.tomorrowssolutionsllc.com).*