Chapter 8
Report Variables

Variables are essential to developing software. It would be impossible to create a practical application without them. Reports are no different! There are many situations when variables are essential to developing reports. You might need a variable to count the number of lines on a page or to track the total sales amount for each customer. Perhaps, you need to count the number of customers for each Sales Rep. Whatever the reason, the Visual FoxPro Report Writer provides a mechanism for creating and maintaining variables within a report.

The Visual FoxPro Report Writer has the ability to use *built-in* Report Variables. Some developers avoid this feature at all costs and adhere to the practice of preparing the data, including calculations, prior to running the report. The reasoning behind this practice is that your code is more flexible to other forms of output. Once the data has been prepared, it's a quick change to send the results to a spreadsheet instead of Visual FoxPro's report engine. If you have variables and calculations in your report, the change becomes much more complex.

That said, I still use Report Variables on many of the reports I design. Sometimes, I prepare the data in advance, but not always. I just can't bring myself to justify building a temporary cursor of 50,000 records when it contains the exact same data as the original table and one additional field that is the result of a simple count or sum function. I know FoxPro is fast, but one of my main concerns when I develop software is maximizing performance for my users, and making them wait while that cursor is built just rubs me the wrong way. So it just depends on the situation and needs of the report I'm designing as to which method I use.

Report Variables are similar to other variables in Visual FoxPro in that they can contain data of different types such as character, numeric, or dates. They also have a *scope*, which is public. This means you can access a VFP Report Variable from outside of the report, such as a UDF or a method call made by the report.

There are two main reasons for using Report Variables. The first is to take advantage of having the Report Writer perform one of its built-in calculation functions. You simply define a Report Variable, set it up as a calculated variable, and that's it. The Report Writer takes care of processing the variable for each record, such as incrementing a Count variable or totaling a Sum variable. The second reason for using Report Variables is to extend the built-in calculation functions on Report objects. See a pattern here? Report Variables are most often used for performing calculations.

Creating variables

Report Variables are created through the Report Variables dialog (shown in **Figure 1**), which can be displayed by selecting Report | Variables… from the main VFP Menu bar. The Variables list box in the upper left corner of this dialog lists all the Report Variables defined for this report. As you select an item from the list, the rest of the information on the dialog refreshes to reflect the properties of the selected Report Variable.

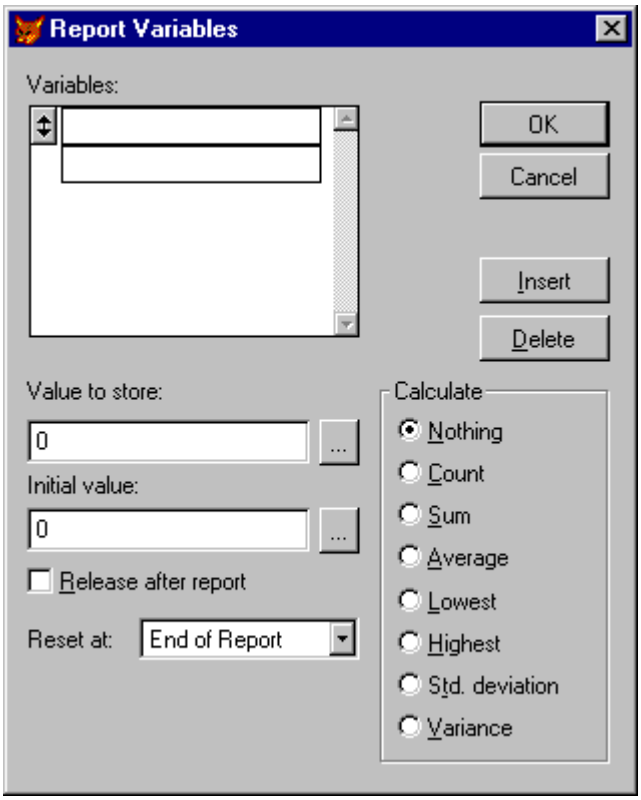

***Figure 1**. Use the Report Variables dialog to create variables used within a report.*

Variable name

To create the first Report Variable, enter a name in the first box of the Variables list box. To create the second Report Variable, enter the name in the second box of the Variables list box. Each time you add a new Report Variable, an additional box is added in the list box so there's always an empty box to enter a new Report Variable.

The rules for naming a Report Variable are the same rules for naming any other VFP variable and are as follows.

- Use only letters, underscores, and numbers.
- Use 1-128 characters.
- Begin the name with a character or underscore.
- Avoid Visual FoxPro reserved words.

My Tech Editor, Dave Aring, brought up a very good point in regards to naming Report Variables. We're all accustomed to the *standard naming conventions* of prefixing variables with a character to identify the scope. For example, global variables are prefixed with "g",

private variables are prefixed with "p", local variables are prefixed with "l", and parameters are prefixed with "t". So shouldn't we prefix Report Variables with something to identify them as Report Variables? Thanks, Dave! That's a great suggestion and from now on, I'm going to start prefixing my Report Variables with "r". Of course, don't hold me to this as you read through the rest of the book—old habits are hard to break.

Insert and Delete

To insert a new Report Variable in a position other than the end of the list, position the cursor in the list box and then select the Insert button. This moves the current item and any subsequent items down and inserts a new item at the cursor position.

To delete an existing Report Variable, position the cursor on the Report Variable and select the Delete button.

Value to store

The Value to store text box is where you assign a value to the Report Variable by entering any valid VFP expression. As each record is processed by the Report Writer, the expression is evaluated and the results are assigned to the Report Variable.

The expression can be as simple as static information such as *"Fox Rocks!"*, a field such as *MyTable.MyField*, a complex expression such as *INT((MyField + 20) / 100)*, or it can even use UDFs or method calls such as *MyUDF(MyField)*. You may also select the ellipse button (…) to the right of the Value to store text box to invoke the Expression Builder dialog to help you enter a more complex expression.

Release after report

The "Release after report" check box is *supposed* to determine whether or not the Report Variable is released when the report is done. Remember, Report Variables are defined as public, which means they stay in memory until CLEAR ALL, CLEAR MEMORY, RELEASE <variable>, or RELEASE ALL is executed. However, the VFP Report Writer has a bug and never bothers to release the Report Variables.

The VFP Report Writer ignores the "Release after report" check box and never releases the Report Variables. If you intend to reference the Report Variables after the report is finished, make sure this option is not checked (just in case this bug is fixed in a future version of Visual FoxPro). For now, the only foolproof way to release Report Variables is to use the RELEASE command after calling the report.

Using calculations

Report Variables have seven built-in mathematical calculation options: Count, Sum, Average, Lowest, Highest, Std. deviation (standard deviation), and Variance. The calculations allow you to process data from all the detail records and print information such as subtotals for each customer or grand totals at the end of the report.

The value of the Report Variable is primed to the *Initial value* when the report begins. Then, as each record is processed, the *Value to store* expression is evaluated and used in the

calculation. When the Report Writer reaches the *reset* point defined for the calculation, the value of the Report Variable is reset to the *Initial value*. Processing then resumes with the next record, building the calculation again. Whenever a calculated Report Variable prints on a report, it's the value of the calculation and *not* the value of the Value to store expression that prints. This fact often confuses developers.

Calculate

The first step in telling the Report Writer to apply an internal calculation to this variable is to select one of the following options from the Calculate option group.

- **Count:** This option counts the number of times this Report Variable is processed—in other words, it counts the number of records processed. The value of the expression defined in the Value to store text box is immaterial to the results of this count. However, the Initial value is very important to the results, as the count begins with this value.
- **Sum:** This option keeps a running total based on the results of the Value to store expression. As each record is processed, the Report Writer evaluates the expression and adds the results to the Report Variable. If the result of the expression does not return a numeric value, this Report Variable is set to .F.
- **Average:** This option performs an *arithmetic mean* (average) based on the Value to store expression. As each record is processed, the Report Writer keeps a separate running total based on the Value to store expression. Then, for the value of this Report Variable, it divides the accumulative total by the number of records processed to obtain the average. If a currency field is used in the Value to store expression, the results are returned with a precision of four decimal places. Otherwise, two decimal places are returned. If the Value to store expression returns a non-numeric value, this Report Variable is set to .F.
- **Lowest:** This option keeps track of what the lowest value of the Value to store expression is for all the records processed so far. However, the *Initial value* is also taken into account and can really skew the results. For example, if the data contains all positive numbers greater than zero, the *initial value* of zero is always less than any of the data. You must remember to set the Initial value to at least one more than the highest value you expect from the Value to store expression of all the records. Note that non-numeric data types (such as character and date) can be returned from the Value to store expression.
- **Highest:** This option keeps track of what the highest value of the Value to store expression is for all the records processed so far. However, as with the Lowest option, the *Initial value* is also taken into account. So, again, you must remember to set the Initial value to something lower than the lowest value that results from the Value to store expression of all the records. And don't be so quick to set it to zero and call it good. If the Value to store expression returns any negative values, you have to set an Initial value to a negative number less than any of the negative numbers returned by the Value to store expression. Note that non-numeric data types (such as character and date) can be returned from the Value to store expression.

- **Std. Deviation:** This option can be used to calculate the square root of the variance (described next). If the Value to store expression returns a non-numeric value, this Report Variable is set to .F.
- **Variance:** This option can be used to measure how spread out a distribution is. It's computed by taking the average squared deviation of each number from its mean. So what does that mean in English? I don't know! How about I explain it in terms we all know—FoxPro. The following code prints the Variance and Standard Deviation for the series of numbers 1-10. The code can be found in a program called Variance.PRG and is included in the source code available with this book. The results are shown in **Figure 2**. If the Value to store expression returns a non-numeric value, this Report Variable is set to .F.

```
*-- Variance and Standard Deviation (calculated manually)

*-- Prime
nTotal = 0

*-- Loop thru the values 1-10
FOR n = 1 to 10

  *-- What's the average so far
  nTotal = nTotal + n
  nAverage = nTotal / n

  *-- Do another loop
  nNewTotal = 0
  FOR x = 1 TO n
    nNewTotal = nNewTotal + (x - nAverage)^2
  ENDFOR

  *-- Calculate the variance and standard deviation
  nVariance = nNewTotal / n
  nDeviation = SQRT(nVariance)

  *-- Print the variance and standard deviation
  ? '  This value: ', n, ;
    '       Variance: ', nVariance, ;
    '       Deviation: ', nDeviation

ENDFOR
```

This value:	1	Variance:	0.0000	Deviation:	0.0000
This value:	2	Variance:	0.2500	Deviation:	0.5000
This value:	3	Variance:	0.6667	Deviation:	0.8165
This value:	4	Variance:	1.2500	Deviation:	1.1180
This value:	5	Variance:	2.0000	Deviation:	1.4142
This value:	6	Variance:	2.9167	Deviation:	1.7078
This value:	7	Variance:	4.0000	Deviation:	2.0000
This value:	8	Variance:	5.2500	Deviation:	2.2913
This value:	9	Variance:	6.6667	Deviation:	2.5820
This value:	10	Variance:	8.2500	Deviation:	2.8723

***Figure 2**. This example shows the results of the Variance and Standard Deviation of a series of numbers (1-10).*

Initial value

The VFP Report Writer evaluates the *Initial value* at the beginning of the report and uses the results to prime the variable. It also resets the Report Variable to the Initial value whenever a *reset* point is reached. The default Initial value is zero.

When using calculations, it's very important to set this value to a meaningful value. As mentioned previously, when using some calculations such as Lowest and Highest, the Initial value is very important to the results of the calculation. Your results might be skewed if you're not careful in choosing an appropriate Initial value.

It's also very important to make sure the results of the Initial value is of the same data type as the results you desire from the calculation. Very strange results can happen when the data type of the Initial value is not as expected, especially when dates are used. For example, if you set the Value to store expression to a numeric data type, set the Initial value expression to a date data type, and set the calculation to Count, the results returned to this Report Variable is the date in the Initial value for *every* record. Or even stranger, change the Value to store expression to a currency field. This time the Julian Day Number of the date in the Initial value is returned to this Report Variable for *every* record. Not exactly the *Count* you were expecting!

Reset at

The Reset at drop-down combo box is used to tell the Report Writer *when* to reset the Report Variable to the Initial value. The default is *End of Report*. The following options are available.

- **End of Report:** The Report Variable is only primed once at the beginning of the report.
- **End of Page:** The Report Variable is reset at the end of each page. The exact point of the reset occurs after all the bands on the current page print and before any bands on the new page print. This means that if you use the Report Variable in the Page Footer band, it still contains the value of the last detail record on the page. If you use the Report Variable in the Page Header band, it contains the value returned by the Initial value.
- **End of Column:** The Report Variable is reset at the end of each column. This option is only available when multiple column sets have been set up for the report. This means that if you use the Report Variable in the Column Footer band, it still contains the value of the last detail record for that column. If you use the Report Variable in the Column Header band, it contains the value returned by the Initial value.
- **<Data Grouping>:** In addition to the three standard options, an additional Reset at option is available for each Data Grouping defined on the report. This allows you to reset the Report Variable each time a new value is encountered for the Data Group. This means that if you use the Report Variable in the Group *Footer* band, it still contains the value of the last detail record for that group. If you use the Report Variable in the Group *Header* band, it contains the value returned by the Initial value.

Using Report Variables

Now that you're able to create Report Variables, you're probably wondering *how* to use them. Well, the answer is, "Just the same as any other variable." You can use the Report Variable by itself as the expression of a Field object. You can also use the Report Variable within another Report Variable or complex expression of a Field object. You can even reference the Report Variable from within UDFs and method calls made by the report. The sample report shown in **Figure 3** uses several different Report Variables. The report definition is included in the source code available with this book, and is called Variables3.FRX.

03/17/2002 **Order Freight - Summarized by Month** Page 1

Order Date	Count	Order #	Country	All Freight	USA Freight
05/09/1992	1	1	Italy	0.00	
05/12/1992	2	2	Canada	79.45	
05/13/1992	3	3	Sweden	36.18	
05/14/1992	4	4	Denmark	18.59	
05/15/1992	5	5	Denmark	20.12	
05/19/1992	6	6	Finland	4.13	
05/20/1992	7	7	Italy	3.62	
05/21/1992	8	8	Germany	36.19	
05/22/1992	9	9	Portugal	74.22	
05/23/1992	10	10	UK	49.21	
05/27/1992	11	11	Denmark	3.01	
05/28/1992	12	12	Brazil	31.54	
05/29/1992	13	13	Venezuela	102.59	
05/30/1992	14	14	Switzerland	50.87	
			Subtotal for May 1992:	**509.72**	**0.00**
06/02/1992	1	15	Venezuela	17.67	
06/04/1992	2	16	Austria	22.10	
06/05/1992	3	17	France	113.01	
06/06/1992	4	18	France	111.81	
06/09/1992	5	19	USA	65.46	65.46
06/10/1992	6	20	Italy	2.42	
06/12/1992	7	21	France	27.51	
06/13/1992	8	22	Austria	75.17	
06/16/1992	9	23	France	46.00	
06/17/1992	10	24	Germany	66.87	
06/18/1992	11	25	USA	5.19	5.19
06/20/1992	12	26	USA	3.32	3.32
06/23/1992	13	27	Germany	1.34	
06/24/1992	14	28	Austria	100.13	
06/25/1992	15	29	Mexico	46.86	
06/26/1992	16	30	Norway	6.72	
06/30/1992	17	31	Venezuela	26.49	
			Subtotal for June 1992:	**738.07**	**73.97**
			Report Total	**79,202.21**	**16,032.82**

Report includes 1079 orders from 36 different months

***Figure 3**. This report uses Report Variables to count the orders per month, to print the USA Freight column, to count the total orders for the entire report, and to count the number of months on the report.*

Simple Report Variables

The report shown in Figure 3 contains a few simple Report Variables to count the number of orders per month and the total number of orders for the entire report. I started by creating a Report Variable called *nCountMonth* that has a Value to store of *1*, an Initial value of *0*, and the *Count* option selected. I set the Reset at value to my Data Grouping so it resets the count for each different month. I then added a Field object to the report and used *nCountMonth* as the expression as seen in the Count column.

Next, I wanted to count the total number of orders of the entire report. I did this by adding another Report Variable called *nCountAll*, which is similar to *nCountMonth.* The only difference is that I set it to reset at *End of Report*. I then used the *nCountAll* variable in the expression of a Field object in the Report Summary band (along with some text and another variable, which I'll describe in a moment).

Conditional Report Variables

Have you ever needed to create a column that only includes certain records *and* has a summary calculation based on those records (as I did with the *USA Freight* column shown in Figure 3)? At first, you might think, "This is easy. I'll use the Print When expression." But you'd only be half right. Why? The reason is that the Print When expression only controls *when* the object prints. It does *not* have anything to do with whether or not the record participates in the calculation. Each and every record processed by the Report Writer causes the Report Variables to be evaluated, including calculations, regardless of whether or not those Report Variables happen to be used by Field objects with a Print When expression.

So now what? Well, to create the USA Freight column, start by creating a Report Variable called *nUSAFreight.* For the Value to store expression enter *IIF(ship_to_country = 'USA', freight, 0)*. For the Calculation option, select *Nothing*. For each record processed, this variable contains zero if it's not USA; otherwise, it contains the freight amount.

Now that you have a Report Variable that only contains a freight amount if the order is for the USA, add a Field object to the Detail band of the report and enter *nUSAFreight* as the expression.

To add a little more finesse to the report, you can also enter *ship_to_country = 'USA'* as the Print When expression of this object so it suppresses all non-USA orders. This is better than selecting the "Blank if zero" option because it helps the user distinguish between USA orders that don't have any freight vs. non-USA orders.

To add the subtotals and totals for this USA Freight column, copy the Field object from the Detail band into the Group Footer and Summary bands. You could also add a new Field object manually, but hey, why do more work than necessary! Once you have Field objects in the Group Footer and Summary bands with *nUSAFreight* as the expression, edit them to set the Calculation option to SUM and the Reset at option to the Data Group and End of Report, respectively. That's it. Nothing more is needed. Since you already told the Report Variable to include the freight amount *only* if the record is for USA, you don't need to do anything more to the SUM. It's already taken care of.

Counting Data Groups

The final line of the report shown in Figure 3 prints the total number of orders *and* it prints the total number of different months encountered. The total number of orders is easy and I just

explained how to do it with a Report Variable using the Count option. So how did I count the total number of months encountered? This takes a little bit of ingenuity, but once you know the secret, it's easy!

The first step is to create a Report Variable (I called mine *nTotalMonths*). Set the Value to store to *0*, the Initial value to *0*, the Calculate option to *Sum*, and the Reset at option to *End of Report*.

Right about now you're saying, "That's stupid. I have a Report Variable that sums nothing. How is that going to work?" Bear with me and it will all make sense. Earlier, I mentioned that Report Variables are public variables. Here's where you can take advantage of this and manipulate the variable yourself with the _VFP.SetVar() function. Just enter the following as the On Exit expression of the Data Group Footer.

```
_VFP.SetVar("nTotalMonths", nTotalMonths + 1)
```

That's it! Whenever the VFP Report Writer encounters a new value for the Data Group, it executes this line of code, which increases the nTotalMonths Report Variable by 1. And remember that you defined the Report Variable as SUM with a Value to store of zero. This means that as each record is processed, the Report Writer does just what you told it to do—it adds zero to the Report Variable and doesn't reset it until the end of the report. Therefore, it maintains your manipulations to the Report Variable without making any of its own.

So, you haven't made the leap to Visual FoxPro and you have no idea what I mean when I say "On Exit expression." Well, once again, FoxPro provides several different ways to do the same thing, so you're not out of luck. Create the following UDF and then add a Field object to the Group Footer band of the report. Enter IncrMonths() *as the expression for the Field object. Problem solved!*

```
FUNCTION IncrMonths
nTotalMonths = nTotalMonths + 1
RETURN ""
```

Note: I consciously put the code to increment the counter in the Group Footer band and not the Group Header band. The reason is that the Data Group can be defined to repeat the Group Header on each page. If this were the case, the variable would get incremented each *time the Group Header printed, which may be more than once for each different month.*

Understanding the process

To fully understand Report Variables you need to understand exactly when the Report Variable is evaluated, when it's reset, and how it relates to other Report Variables. Otherwise, you may struggle with trying to get them to work the way you want.

Order of Report Variables

When multiple Report Variables are defined on a report, they're processed in the order they appear in the Variables list box. This is important if one Report Variable references another Report Variable. You can use the Mover button to the left of each item in the list box to reposition the Report Variables.

Evaluating a Report Variable

Report Variables are evaluated by the VFP Report Writer just prior to the current record being printed in the Detail band. The sample report shown in **Figure 4** demonstrates exactly when a Report Variable is processed. The first column shows which band is printed. The second column shows the value of a Report Variable (defined as Count). The third column shows the current record number. This report, called Variable1.FRX, is included in the source code available with this book.

	Report Variable (Count reset at End of Report)	RECNO()	
Page Header	0	1	
Group Header	0	1	
Detail	1	1	
Detail	2	2	
Detail	3	3	
Group Footer	3	3	
Group Header	3	4	
Detail	4	4	
Detail	5	5	
Detail	6	6	
Page Footer	6	6	Page 1

***Figure 4**. This sample report demonstrates when Report Variables are evaluated.*

Notice the Report Variable and the RECNO() value don't always match. This is the part that can bite you if you're not expecting it. In Page Header and Group Header bands, the

RECNO() reflects the record that is about to be printed next because the record pointer has already been moved. However, the Report Variable hasn't been processed yet, so it still reflects the value from the previously printed record.

Resetting a Report Variable

You already know that when you define a Report Variable, you tell it what the Reset point is. In other words, you tell it when to reset to its Initial value. But exactly *when* the variable is reset is important, especially when compared to when the Report Variable is evaluated. For example, see the sample report shown in **Figure 5**, which has a Report Variable defined as Count and Reset at Data Group. This report, called Variable2.FRX, is included in the source code available with this book.

	Report Variable (Count reset at Data Group 1)	RECNO()	
Page Header	0	1	
Group Header	0	1	
Detail	1	1	
Detail	2	2	
Detail	3	3	
Group Footer	3	3	
Group Header	0	4	
Detail	1	4	
Detail	2	5	
Detail	3	6	
Page Footer	3	6	Page 1

***Figure 5**. This sample report demonstrates when Report Variables are reset.*

You should notice that although the Report Variable is reset to zero after it finishes the Data Group, it doesn't get evaluated again until the next Detail band is processed. This means that if you reference the Report Variable in the Page Header or Group Header band, the value

it contains is the Initial value. So even though the record pointer is on the next record, the Report Variable hasn't been processed yet.

Mission impossible

As I mentioned at the beginning of this chapter, it's impossible to create an application without using variables, and reports are no different. There are many different situations that warrant the use of Report Variables. Understanding exactly how the Visual FoxPro Report Writer processes these variables helps you write the reports as easily as possible.