

# Cross-Tabulation & Related Reports

## Part II: More Reports

by Michael P. Goldsmith

Cross-tabulation reports are very common in business. A standard cross tabulation report is a spreadsheet-like report that provides counts of the number of records that fall into a category defined by the intersection of values in two fields in a database. This article demonstrates techniques to produce two types of reports. The first report is shown in Figure 1.

Figure 1: Cross Tabulation Report with Percents

|          | Number<br>Engineers | Number<br>Contacts | %<br>Contacted | Number<br>Sales | %<br>Sales |
|----------|---------------------|--------------------|----------------|-----------------|------------|
| Denver   | 32                  | 24                 | 75%            | 13              | 41%        |
| Houston  | 35                  | 19                 | 54%            | 12              | 36%        |
| Seattle  | 24                  | 16                 | 66%            | 15              | 45%        |
| Portland | 33                  | 23                 | 69%            | 14              | 42%        |
| TOTAL:   | 124                 | 82                 | 66%            | 54              | 44%        |

The second report uses the techniques discussed in Part I of this article but the data is loaded into a 4D Calc spreadsheet (Figure 2). The example database uses a Mac OS demonstration version of 4D Calc. The techniques used to produce the 4D Calc report resulted from the needs of a customer to have both printed reports and to load the report data into a spreadsheet so he could perform “what-if” analysis on the data.

Figure 2: 4D Calc Spreadsheet Report

|   |          | No. Sales | 1 to 7,000 | 7,001 to 10,000 | Over 10,000 | Total |
|---|----------|-----------|------------|-----------------|-------------|-------|
| 4 | Colorado | 10        | 4          | 0               | 7           | 21    |
| 5 | Houston  | 20        | 3          | 3               | 6           | 32    |
| 6 | Seattle  | 9         | 8          | 3               | 9           | 29    |
| 7 | Portland | 15        | 6          | 4               | 4           | 29    |
| 8 | Total:   | 54        | 21         | 12              | 26          | 124   |

The examples can produce many different reports by varying just the parameters passed to the procedures that actually produce the reports. Since it is quite common for managers to want to see many similar reports based on data in the same file or files, the procedures are written to allow many reports to be produced with minimal coding changes. For clarity, the examples are based on a simple file structure containing some basic sales related information. (Figure 3)

Figure 3: Table Structure

| Companies   |   |
|-------------|---|
| Company     | ⓧ |
| City        | ⓧ |
| State       | ⓧ |
| Salesperson | ⓧ |
| Contacted   | ⓧ |
| Sales       |   |

## Principles

In order to make the process as fast as possible, each record will be read only once. This means that all processing for that record must be performed at the same time. Second, the information for each report will be stored in a unique two-dimensional text array. Two-dimensional arrays in 4D originally used the format array{Column}{Row}. In 4D v3 the format was changed to array {Row}{Column}. For the purpose of these examples, I am maintaining the original format so I will use array{\$col}{\$row}.

Third, pointers will be used extensively in order to make the procedures independent of the fields and arrays that are being processed. This allows the same set of procedures to be used many times. The procedures are also easily modified to produce variations of the reports shown in the examples. Lastly, both types of reports will be built during the single pass through the data.

## Implementation

The techniques for initializing, filling, and sorting the two dimensional arrays were discussed in part I of this article which appeared in the March 2001 issue of Dimensions. The main method is shown below.

### CRT\_PartIIDemo

```

`PM: CRT_PartIIDemo Mike Goldsmith
`This method will produce a series of Cross-tabulation Reports each based on
`different Reporting Fields. One of the reports will be created in a 4D Calc sheet
C_BOOLEAN($MGC010601)
C_LONGINT($rec; $RIS)
ARRAY REAL(arTotals; 6) `Declaring totals array
CRT_Init2DArrs (6; 0) `initializes the arrays
ALL RECORDS([Companies]) ` You may allow the user to conduct a Query instead
$RIS: =Records in selection([Companies])
ThermoInit ($RIS; " Processing records. . .")
For ($rec; 1; $RIS)
  ThermoUpdate ($rec)
  CRT_LoadArrays (->atTalliesA; ->[Companies]City; ->[Companies]Sales)
  CRT_LoadPrCntArrays (->atTalliesC; ->[Companies]City; ->[Companies]Contacted;
->[Companies]Sales)
  NEXT RECORD([Companies])
End for `($rec; 1; $RIS)
ThermoClose
CRT_Sort2D (->atTalliesA)
CRT_Sort2D (->atTalliesC)
`Cross tab printed report with percentages
CRT_ArrayCalculations (->atTalliesC; "B")
CRT_Print (->atTalliesC; "B"; "Contacts and Sales by City")
`cross-tab report loaded into 4D Calc
calc_Demonstration (->atTalliesA)
CRT_Init2DArrs (0; 0)

```

**CRT\_LoadPrCntArrays** is a modification of **CRT\_LoadArrays**. The only difference is that in **CRT\_LoadPrCntArrays** not all of the columns are filled on the initial loop since some of the columns will contain percents that are calculated separately. The method, **CRT\_ArrayCalculations**, that calculates the totals was modified to handle reports with just counts and reports with percentages.

## CRT\_ArrayCalculations

```

`PM: CRT_ArrayCalculations
`Mike Goldsmith - 5/22/01 11:00 AM
`$1 : Pointer to array; $2 : report type
`Sums each column for both types
`Calculates percents for type B report
C_POINTER($ArrPtr; $1)
C_STRING(2; $suffix; $2)
C_LONGINT($Numrows; $Numcols; $row; $col)
ARRAY REAL(arTotals; 0) `clearing totals array
ARRAY REAL(arTotals; 6) `declaring totals array
$ArrPtr: =$1
$suffix: =$2
$Numrows: =Size of array($ArrPtr->{1}) ` returns the number of rows
$Numcols: =Size of array($ArrPtr->) ` returns the number of columns
Case of
: ($suffix="A")` handles straight counts
For ($col; 2; $Numcols) `loop for each column starting with column 2
  For ($row; 1; $Numrows) `loop for each row
    arTotals{$col}: =arTotals{$col}+Num($ArrPtr->{$col}{$row}) ` sums each column
  End for `($row; 1; $Numrows) loop for each row
End for `($col; 2; $Numcols) loop for each column starting with column 2
: ($suffix="B")` handles reports with percentages
For ($col; 2; $Numcols) `loop for each column starting with column 2
  If (($col#4) & ($col#6)) ` columns 4 & 6 do not need to be totaled
    For ($row; 1; $Numrows) `loop for each row
      arTotals{$col}: =arTotals{$col}+Num($ArrPtr->{$col}{$row}) ` sums each column
    End for `($row; 1; $Numrows) loop for each row
  End if `(($col#4) & ($col#6)) ` Columns 4 & 6 do not need to be totaled
End for `($col; 2; $Numcols) loop for each column starting with column 2
For ($row; 1; $Numrows) ` calculates the percentages for each row for columns 4 & 6
  $ArrPtr->{4}{$row}: =String(Round((Num($ArrPtr->{3}{$row})/Num($ArrPtr->{2}{$row}))*100; 0))
  $ArrPtr->{6}{$row}: =String(Round((Num($ArrPtr->{5}{$row})/Num($ArrPtr->{2}{$row}))*100; 0))
End for `($row; 1; $Numrows)
arTotals{4}: =Round((arTotals{3}/arTotals{2}*100); 0)
arTotals{6}: =Round((arTotals{5}/arTotals{2}*100); 0)
End case

```

The difference in the calculations needed by the two types of reports is handled by passing a report type to the method. Type "A" calls for simple counts. Type "B" handles the reports with percentages. In that section, I hard-coded the column numbers for the columns that will contain the percents in order to make the code more readable. Since the array used to hold all the data is a string array, in order to do the calculations, I needed to convert the data to numeric values, do the calculations, and then convert the values back to strings.

```

$ArrPtr->{4}{$row}: =String(Round((Num($ArrPtr->{3}{$row})/Num($ArrPtr->{2}{$row}))*100; 0))

```

handles this task in one line of code. After the calculations have been completed, the report is ready to be printed. This is handled by **CRT\_Print**.

## CRT\_Print

```

`PM: CRT_Print Mike Goldsmith
`prints cross-tabulation & related reports called from the main method
`$1: pointer to Tally array; $2: header suffix; $3: report title
`NOTE: I am using array{column}[row] for these methods
C_POINTER($ArrPtr; $1)
C_STRING(2; $suffix; $2)
C_TEXT($title; $3; $header)
C_LONGINT($Numrows; $Numcols; $row; $col)
$ArrPtr:=$1
$suffix:=$2
$title:=$3
$Numrows:=Size of array($ArrPtr->{1}) ` returns the number of rows
$Numcols:=Size of array($ArrPtr->) ` returns the number of columns
PRINT SETTINGS ` called when using PRINT LAYOUT
If (OK=1)
  tTitle:=$title
  $header:="CRT_Header"+$2 `adds the report type to the header name to call the correct header
  PRINT FORM([Companies]; $header)
  For ($row; 1; $Numrows) `loop for each row
    apPointer{1}->:=$ArrPtr->{1}{$row}
    For ($col; 2; $Numcols) `loop for each column starting with column 2
      If (($2="B") & (($col=4) | ($col=6))) `` column contains a percentage
        apPointer{$col}->:=String(Num($ArrPtr->{$col}{$row}); "#, ##0")+ "%"
      Else
        apPointer{$col}->:=String(Num($ArrPtr->{$col}{$row}); "#, ##0")
      End if ` If (($2="B") & (($col=4) | ($col=6))) `` column contains a percentage
    End for `($col; 2; $Numcols) loop for each column starting with column 2
    PRINT FORM([Companies]; "CRT_Details")
  End for `($row; 1; $Numrows) loop for each row
  For ($col; 1; $Numcols)
    apPointer{$col}->:="" ` clears the layout objects
  End for
  PRINT FORM([Companies]; "CRT_Details") ` prints a blank line
  apPointer{1}->:="TOTALS:"
  For ($col; 2; $Numcols)
    If (($2="B") & (($col=4) | ($col=6))) ` column contains a percentage
      apPointer{$col}->:=String(arTotals{$col}; "#, ##0")+ "%"
    Else
      apPointer{$col}->:=String(arTotals{$col}; "#, ##0")
    End if (($2="B") & (($col=4) | ($col=6))) ` column contains a percentage
  End for `($col; 2; $Numcols)
  PRINT FORM([Companies]; "CRT_Details") ` prints the totals
  PAGE BREAK ` Forces printing of the last page
End if `(OK=1)

```

This method was modified slightly to handle the percentages. An **If – End if** clause was added to allow the adding of the “%” symbol to the percentages before they are displayed in the report.

The method **calc\_Demonstration** contains the 4D Calc routines.

It is called from **CRT\_PartIIDemo** and passed a pointer to the array **atTalliesA**.

## calc\_Demonstration

```

`Method: calc_Demonstration - Mike Goldsmith
`Loads report title, column titles, and data into cells
C_BOOLEAN($MGU010601)
C_TEXT($Title)
C_LONGINT($HeaderRow; $StartData; $startTotalRow; $NumColumns)
C_POINTER($1; $ArrayPtr)
$ArrayPtr: =$1 ` use local variable for better readability
`report data stored in preference table to allow for user modification
QUERY([Preferences]; [Preferences]Report="CalcDemo")
$Title: =[Preferences]ReprtHeadText
$HeaderRow: =[Preferences]HeaderRowNum

`row and column information is calculated
$StartData: =$HeaderRow+1
$startTotalRow: =$StartData+Size of array($ArrayPtr->{1})
$NumColumns: =Size of array($ArrayPtr->)

`opens an external window for 4D Calc
calc_Window (50; 50; 600; 450; $Title)

`loads the column titles
calc_Coltitles ($HeaderRow; "No Sales"; "1 to 7, 500"; "7, 501 to 15, 000"; "Over 15, 000"; "Total")

` loads the arrays into the cells
calc_LdRepArrs ($StartData; $ArrayPtr)

` calculates the sums of each column
calc_Totals ($NumColumns; $StartData; $startTotalRow)

` loads the headers and footers and Formats the spreadsheet
calc_HeadFoot ($Title)
`formats the spreadsheet
calc_Format ($NumColumns)

```

The Report Title and the Header row number are stored in a preference table record so they can be modified by the users. These values are retrieved at the top of the method. The starting row for the data and the row for the column totals are calculated next and stored in local variables. The number of columns in the report is determined by finding the size of the two-dimensional array holding the data. The **calc\_Window** method is then called.

## calc\_Window

```

`PM: calc_Window - Mike Goldsmith
`$1 : Left Coord $2 : Top Coord
`$3 : Right Coord $4 : Bottom Coord
`$5 : Window Title
C_BOOLEAN($MGU010601)
$left: =$1
$right: =$3
$stop: =$2
$bottom: =$4
$title: =$5
$type: =Plain window
C_LONGINT(lcalcSheet; $1; $2; $3; $4; $left; $stop; $right; $bottom)
C_STRING(40; $5; $title)
lcalcSheet: =Open external window($left; $stop; $right; $bottom; $type; $title; "_4D Calc")

```

An external window is opened for 4D Calc using the 4D command **Open external window**. The variable **lcalcSheet** is assigned the window reference number, which is used to identify the 4D Calc Window in the rest of the methods. The column headers are loaded into the correct spreadsheet cells in **calc\_Coltitles**.

## calc\_Coltitles

```

`PM: calc_Coltitles - Mike Goldsmith
`Generic procedure : Headings passed as strings
`Builds Headings row for 4D calc report
` The first paramter is the row number for the column titles
C_BOOLEAN($MGU010601)
C_LONGINT($i; $1; $row; $columns)
C_STRING(40; $1)
C_STRING(40; ${2})
$row: =$1
For ($columns; 2; Count parameters)
  SP SET CELL STRING (lcalcSheet; SP Cell ($columns; $row); ${$columns})
End for

```

The 4D Calc command **SP SET CELL STRING(Area; Cell; String)** assigns the **String** to the **Cell**. **Area** is the 4D Calc area and in these methods it is identified by the window reference number stored in **lcalcSheet**. The cell value is an internal reference number. **SP Cell(Column;Row)** is a function that takes the column and row numbers and returns the internal reference number for that cell. You can use one line of code to handle many cells by using variables for the column and row and placing the command in a loop. In order to call all the strings that were passed to this method, parameter indirection was used.

The data in the array will next be loaded into Cells using **calc\_LdRepArns**

## calc\_LdRepArRs

```

`PM: calc_LdRepArRs - Mike Goldsmith
`Loads the Arrays into the appropriate columns in the Spreadsheet
`$1 Starting row; $2: =Pointer to 2D array
C_BOOLEAN($MGU010601)
C_LONGINT($1)
C_POINTER($ArrPointer)
$ArrPointer: =$2
$NumColumns: =Size of array($ArrPointer->)
$NumRows: =Size of array($ArrPointer->{1})
For ($Column; 1; $NumColumns)
  SP ARRAY TO CELLS (lcalcSheet; 1; SP Cell ($Column; $1); $ArrPointer-
->{$Column}; $NumRows; 1)
End for

```

The 4D Calc command **SP ARRAY TO CELLS(Area;Axis;StartCell;Array;Numb)** is used to load array data into cells in the spreadsheet. The axis value controls whether the arrays are loaded into columns or rows. The method **calc\_LdRepArRs** uses a 1 to load the arrays into columns.

The next method, **calc\_Totals** builds formulas for the column totals and loads the formulas into the correct cells in the spreadsheet.

## calc\_Totals

```

`PM: calc_Totals - Mike Goldsmith
`Places formulas for totals & averages in correct Cells in the 4D Calc window
`$1 : Number of columns in report; $2: The starting row for the data
`$3: The row for the totals
C_BOOLEAN($MGU010601)
C_LONGINT($1; $Numcolumns; $2; $StartData; $3; $Totalrow; $column)
C_STRING(2; $Columnletter)
C_STRING(20; $Formula)
C_INTEGER($row)
$Numcolumns: =$1
$StartData: =$2
$Totalrow: =$3

SP SET CELL TEXT (lcalcSheet; SP Cell (1; $Totalrow); "Totals: ")
For ($column; 2; $Numcolumns) ` loop through the columns starting with column 2
  $Columnletter: =Char(64+$Column) ` Get the column "letter"
  $Formula: ="=Sum("+ $Columnletter+String($startData)+" "+ $Columnletter+String($Totalrow-1)+")"
  SP SET CELL TEXT (lcalcSheet; SP Cell ($column; $Totalrow); $Formula)
End for `($column; 2; $Numcolumns) ` loop through the columns starting with column 2

```

The command **SP SET CELL TEXT** assigns a text value to **Cell**. If the text value starts with the equals sign ("="), Value will be interpreted as a formula. The first call using the command places the row title "Total:" into a Cell in the first column. The method then builds the formula in a loop so the formula is matched to its column. You can use a similar technique to build any spreadsheet formula needed for your reports.

The next method, **calc\_HeadFoot**, formats the header and footer for the printed report produced by 4D Calc.

## calc\_HeadFoot

```

`Method: calc_HeadFoot Mike Goldsmith
`Formats the header and footer for printing
`$1 : report Title
C_BOOLEAN($MGU010601)
C_STRING(80;$1;$Title)
$title:=$1
`Centers the title in the header
SP HEADER FOOTER (lcalcSheet; 2; $title)
`Left justifies the page number
SP HEADER FOOTER (lcalcSheet; 4; "Page #P")
`Right justifies the date
SP HEADER FOOTER (lcalcSheet; 6; "#d")

```

The 4D Calc command, **SP HEADER FOOTER(Area;Number;Text)**, sets the text of the headers and footers. The **Number** value controls the region to be set. The numbers 1 to 3 set the right, center, and left **Header**. The numbers 4 to 6 set the right, center, and left **Footer**. The **Text** can be any text or codes to insert values generated by 4D Calc. The code #p inserts the current page number and the code #d inserts the current date.

The last method, **d\_CalcFormat**, calls several 4D Calc formatting commands.

## d\_CalcFormat

```

`Method: d_CalcFormat Mike Goldsmith
`Formats Cal Sheet & Footers
`$1 : number of columns to size
C_BOOLEAN($MGU010601)
C_LONGINT($1;$cols)
$cols:=$1
` Formats the cells
SP DEFAULT FONT (lcalcSheet; SP Font number ("Geneva"); 9; 0; 0; 0; -1)
`Sizes the columns to minimum size plus 10 points
SP AUTO SIZE (lcalcSheet; 1; 1; $cols; 10)
`Formats the header and footer for printing
SP DEFAULT FONT (lcalcSheet; SP Font number ("Geneva"); 10; 1; 1; 2)

```

The command **SP DEFAULT FONT(Area;Font;Size;Style;Color;Code;Background Color)**, sets the default font for the part of the spreadsheet indicated by the value of **Code**. If code is 0, then the cells are formatted. If the value is 1, the **Header** and **Footer** are formatted. The command is called twice in this method, once for the **Cells** and once for the **Header** and **Footer**. **SP AUTO SIZE(Area;Axis;Start;Points;)** is used to size the columns. The **Points** value is the number of points wider than the minimum width needed to display the data in each column.

In summary, these techniques provide a means to produce variations of the standard cross-tabulation report. The procedures allow the creation of families of similar reports by just adding new arrays and a few new procedure calls in the main procedure. The code can be easily adapted to produce many types of similar reports. The techniques used to produced the 4D calc report can be used for a large variety of reports, not just cross-tabulation reports.

## Parameter Indirection

Parameters are usually addressed by their numeric value: \$1, \$2, etc. This does not work when you want to pass a variable number of parameters to a method. When you need to have a variable

length parameter list, you can use parameter indirection to manage the parameters. Parameter indirection allows you to refer to a parameter indirectly with a variable. For example, if \$column is equal to 3, then \${column} is equivalent to \$3. Using parameter indirection allows a method to use a variable amount of parameters in a loop.

There is a special syntax for declaring parameters of this type.

```
C_TEXT($1)` declare the first parameter  
C_TEXT(${2})` parameters $2 through $n are all declared.
```

First you declare one parameter directly. Then you use indirection to declare all parameters starting with the number in the declaration.¶

---

## About the author

Michael P. Goldsmith is the president of Small System Solutions, [www.sss4d.com](http://www.sss4d.com), which has been providing systems design, database development, and training services for businesses in the U.S. and Canada since 1982. Small System Solutions also provides web design services. Michael has been working with 4D since 1987, and can be reached at [mike@sss4d.com](mailto:mike@sss4d.com).