

Cross-Tabulation and Related Reports

Part I

by Michael P. Goldsmith, mike@sss4d.com

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. The first column contains the values in one field that will be referred to as the reporting field in this article. The first row, column headings, contains the values of the second, or comparison field. The last column contains the totals for each row and the last row contains the column totals and the grand total. An example showing the number of customers by city for each salesperson might produce a report such as:

	Edmonton	Vancouver	Toronto	Total
Hill	3	5	8	16
Johnson	2	5	7	14
Kellog	6	3	2	11
Total	11	13	17	41

This article demonstrates a technique that produces counts based on ranges of values for the comparison field. The second article will show an example using columns that contain percents as well as a method to copy the arrays into 4D Calc. 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 table structure containing some basic sales related information. (Figure 1).

Figure 1 – Table Structure

Companies	
Company	A
City	A
State	A
Salesperson	A
Contacted	D
Sales	05

Principles

First, 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 first task is to determine what arrays need to be declared. This example creates two reports, so two arrays will be needed. These will be atTalliesA, and atTalliesB. The layout for the report will need six objects, one for each column. These will be tObject1 ... tObject6. An array of pointers will need to be declared. CRT_Init2DArrs shows the procedure used to declare the arrays. The text arrays are declared to have 6 columns and O rows. The rows are added later. The procedure accepts a variable to indicate whether it is to be used for initializing or clearing. This allows the procedure to initialize the pointer array with six elements and to clear it later to zero elements.

CRT_Init2DArrs

```
`PM: CRT_Init2DArrs
`Mike Goldsmith - 3/18/01 4: 45 PM
`Initializes, resizes and clears the arrays
`Called from the Main Method
`A separate array is needed for each report
`$1 is for the number of columns in the 2D array
`$2 is the number of rows in the 2D arrays
C_LONGINT($1; $rows; $i; $2; $cols)
$cols: =$1
$rows: =$2
ARRAY TEXT(atTalliesA; $cols; $rows) ` declares a 2 dimension array with 6 columns & ($rows) rows
ARRAY TEXT(atTalliesB; $cols; $rows)
ARRAY TEXT(atTalliesC; $cols; $rows)
ARRAY POINTER(apPointer; $cols) `initialize the pointer array
If ($cols>0)
  For ($i; 1; 6)
    apPointer{$i}: =Get pointer("tObject"+String($i)) `creates pointers to tObject1 totObject6
  End for `($i; 1; 6)
End if
```

The next task is to create the method CRT_LoadArrays that will load the reporting values and counts into the arrays.

CRT_LoadArrays

```
`PM: CRT_LoadArrays
`Mike Goldsmith - 3/18/01 5: 09 PM
`Loads reporting data and counts into array columns
`called from the Main Procedure
`Note: all parameters are pointers
`$1: Array; $2 Reporting field; $3 Comparison field
`NOTE: I am using array{column}[row] for these methods
C_POINTER($ArrPtr; $ReportPtr; $CompPtr)
C_LONGINT($row; $col)
$ArrPtr: =$1
$ReportPtr: =$2
$CompPtr: =$3
$row: =Find in array($ArrPtr->{1}; $ReportPtr->) ` looks for the reporting field value in array column 1
If ($row=-1) ` the value was not found
  $row: =Size of array($ArrPtr->{1})+1 ` create a new row
  For ($col; 1; Size of array($ArrPtr->))
    INSERT ELEMENT($ArrPtr->{$col}; $row; 1) ` resize each column
  End for `($col; 1; Size of array($ArrPtr->))
  $ArrPtr->{1}{$row}: =$ReportPtr-> ` assign the reporting value to the first column of the new row
End if ` ($row=-1) ` the value was not found
Case of
```

```

: ($CompPtr->=0) ` value is 0
  $ArrPtr->{2}{$row}: =String(Num($ArrPtr->{2}{$row})+1) ` add 1 to the 2nd column
: ($CompPtr-><=7500) ` value is less than or equal to 7, 500
  $ArrPtr->{3}{$row}: =String(Num($ArrPtr->{3}{$row})+1) ` add 1 to the 3rd column
: ($CompPtr-><=15000) `value is less than or equal to 15, 000
  $ArrPtr->{4}{$row}: =String(Num($ArrPtr->{4}{$row})+1) ` add 1 to the 4th column
Else
  $ArrPtr->{5}{$row}: =String(Num($ArrPtr->{5}{$row})+1) ` add 1 to the 5th column
End case
$ArrPtr->{6}{$row}: =String(Num($ArrPtr->{6}{$row})+1) ` add 1 to the 6th column ( row total)

```

The first step assigns the row number of the reporting value in the first column of the array to \$row. If the value is not found, Find in array returns -1. Also a new row must be created, the new row number assigned to \$row, and the first column assigned the new value. Next, the count in the appropriate column must be incremented by one. Since text arrays are used, the code to increment the values might look a little funny. Num(\$ArrPtr->{2}{\$row}) returns the numeric value of the text. A 1 is added to the numeric value, then the String(Num(\$ArrPtr->{2}{\$row})+1) expression converts the number back to a string so it can be assigned to the text array again. The Case of statement compares the value of the comparison field to specific values to determine which column to use. Remember, the Case of statement only performs the operations related to the first True statement that it reaches. Therefore :(\$CompPtr-><=7500) will be true only if \$CompPtr is greater than 0 and less than or equal to 7500 since the first Case of statement has already handled the cases where \$CompPtr is 0. The last line increments the count in the last column, which maintains the row total.

The main method CRT_Main is now ready to be started:

CRT_Main

```

`PM: CRT_Main Mike Goldsmith - 3/26/01
`This method will produce a series of Cross tabulation Reports each based on
`different Reporting Fields
C_LONGINT($rec; $RIS)
CRT_Init2DArrs (6; 0) `initializes the arrays
ALL RECORDS([Companies]) ` You may allow the user to conduct a Query instead
$RIS: =Records in selection([Companies])
Thermolnit ($RIS; " Processing records. . .")
For ($rec; 1; $RIS)
  ThermoUpdate ($rec)
  CRT_LoadArrays (->atTalliesA; ->[Companies]City; ->[Companies]Sales)
  CRT_LoadArrays (->atTalliesB; ->[Companies]Salesperson; ->[Companies]Sales)
  NEXT RECORD([Companies])
End for `($rec; 1; $RIS)
ThermoClose
CRT_Sort2D (->atTalliesA)
CRT_Sort2D (->atTalliesB)
CRT_Print (->atTalliesA; "A"; "Customers in Sales categories by City")
CRT_Print (->atTalliesB; "A"; "Customers in Sales categories by Salesperson")
CRT_Init2DArrs (0; 0)

```

All the records in the file are used, but normally a choice of using all records or creating a selection should be left to the user. A For loop is created that will loop once for each record in the selection. I used ThermoSet, free from DataCraft, to show the user of the progress through the selection. CRT_LoadArrays is called once for each different report to be printed. The array and the fields needed to produce each report are passed to the procedure. In order to produce ten reports based on ten different reporting fields, this procedure would be called ten times instead of the two in this example. The ten arrays would have to be initialized first.

After the loop is finished, the arrays need to be sorted. CRT_Sort2D sorts the arrays based on the first column, which holds the reporting value.

CRT_Sort2D

```

`PM: CRT_Sort2D Mike Goldsmith - 3/25/01

```

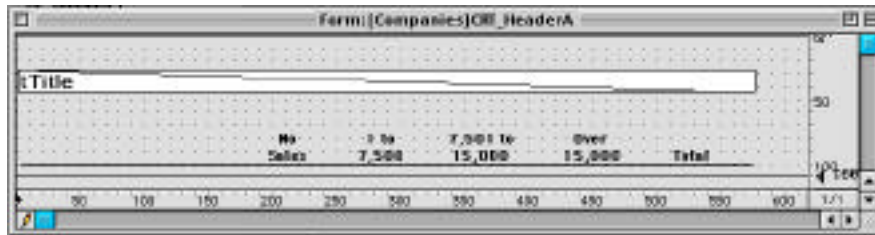
```

`sorts 2D arrays up to 6 columns by treating each column as a separate array
`for larger arrays, add more case statements
`$1 : pointer to the 2D arrays NOTE: I am using array{column}[row] for these methods
C_POINTER($1;$ArrayPtr)
C_LONGINT($cols)
$ArrayPtr:=$1 `use local variable for better readability
$cols:=Size of array($ArrayPtr->) `returns the number of columns(arrays) in the pointer array
Case of
: ($cols=2)
  SORT ARRAY($ArrayPtr->{1}; $ArrayPtr->{2})
: ($cols=3)
  SORT ARRAY($ArrayPtr->{1}; $ArrayPtr->{2}; $ArrayPtr->{3})
: ($cols=4)
  SORT ARRAY($ArrayPtr->{1}; $ArrayPtr->{2}; $ArrayPtr->{3}; $ArrayPtr->{4})
: ($cols=5)
  SORT ARRAY($ArrayPtr->{1}; $ArrayPtr->{2}; $ArrayPtr->{3}; $ArrayPtr->{4}; $ArrayPtr->{5})
: ($cols=6)
  SORT ARRAY($ArrayPtr->{1}; $ArrayPtr->{2}; $ArrayPtr->{3}; $ArrayPtr->{4}; $ArrayPtr->{5}; $ArrayPtr->{6})
End case

```

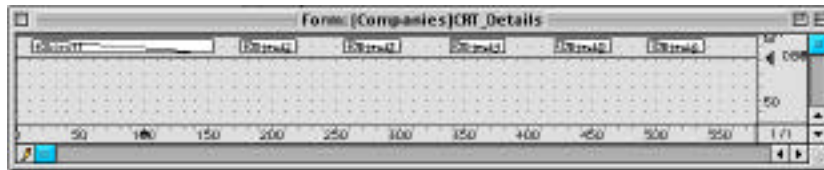
Finally, the report must be printed. The first step is to create the layouts that will be used to print the reports. Two layouts will be needed, one to print the header and one for the detail and total lines. The header layout is shown in Figure 6.

Figure 6– Header Layout



Notice that the D, BO, and F lines are together. The height of the printed header will be 110 points. The detail layout is shown in Figure 7.

Figure 7 – Detail Layout



Each detail line will be 16 points. The project method CRT_Print handles printing the reports:

CRT_Print

```

`PM: CRT_Print Mike Goldsmith - 3/25/01 7:41 PM
`prints the cross tabulation and 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; $col)
ARRAY REAL($arTotals; 6) `Declaring totals array
$ArrPtr:=$1
$suffix:=$2
$title:=$3
PRINT SETTINGS ` called when using PRINT LAYOUT

```

```

If (OK=1)
$Numrows:=Size of array($ArrPtr->{1}) ` returns the number of rows
$Numcols:=Size of array($ArrPtr->) ` returns the number of columns
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; $Numcols) loop for each column starting with column 2
tTitle:=$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; $Numcols) `loop for each column starting with column 2
    apPointer{$col}->:=String(Num($ArrPtr->{$col}{$row}); "#, ##0")
  End for `($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; $Numcols)
  apPointer{$col}->:=String($arTotals{$col}; "#, ##0")
End for
PRINT FORM([Companies]; "CRT_Details") ` prints the totals
PAGE BREAK ` Forces printing of the last page
End if `(OK=1)

```

PRINT LAYOUT is used to print all the elements of the report. The first double For loop is used to find the sum of each column and to assign the values to a Real array with an element for each column. The header is then printed. Next a For loop is created to print each row. Inside the loop the values in each column of the row are assigned to the layout objects using pointers and the layout is printed. After all the rows have been printed, a space is passed to each layout object and the layout is printed to produce a blank line. The totals are then assigned to the layout objects and the layout is printed once more. A form feed is issued to indicate the end of the page and the \$arTotals array is resized to 0 to clear all the values. The last step in the main procedure is to call CRT_Init2DARRs and pass a (0;0) to clear all the arrays. The first finished report is shown below:

Finished Report

Customers in Sales Categories by City

	No Sales	1 to 7,500	7,501 to 15,000	Over 15,000	Total
Execond	17	4	6	4	31
Manchester	16	5	4	5	30
North	14	9	6	4	33
Portsmouth	19	9	9	3	40
TOTALS:	66	27	25	16	134

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 three new procedure calls in the main procedure. The code can be easily adapted to produce many types of similar reports. The next article will examine the techniques used to produce a report with percentages and copy the report values into 4D Calc.

Two-dimensional Arrays

A two-dimensional array stores information in a table of elements of a single type. It has width (columns) and height (rows). The same commands are used to declare a two-dimensional array as are

used in a one-dimensional array except that an additional parameter, row, is added. The declaration for a text array with 5 rows and 6 columns is: ARRAY TEXT(aText;5,6). Each row in a two-dimensional array may be treated separately as a one-dimensional array. Thus, you can apply any of the array commands to a single row. SORT ARRAY(aText(l); ->) sorts the elements in the first row independently of the rest of the rows. Because each row can be treated separately, INSERT ELEMENT or DELETE ELEMENT produces a two-dimensional array with rows of different lengths.

Care is required when using two-dimensional arrays because many of the array commands work differently for one-dimensional arrays. Size of array(aText) would return 5, the number of rows. To find the number of columns in row one, Size of array(aText{1}) would return 6. INSERT ELEMENT(aText;5) would insert a new row at position 5. In order to insert a new column in all the rows of the array, insert a new column in each row separately:

```
For(row; 1; 6)
  INSERT ELEMENT(aText{row}; 5)
End for
```

This will insert a new column at position 5 in each row of the array. A two-dimensional array can not be sorted by itself: the statement SORT ARRAY(aText;>) has no meaning. In order to sort a two-dimensional array, each column must be sorted independently:

```
SORT ARRAY(aTextt[1]; aTextl{2}; aText{3}; aText{4}; >)
```

This statement sorts each row based on row 1.\

Get pointer

The function, Get pointer, procedurally constructs a pointer to an object. The function takes a character string as an argument and returns a pointer to the object of the same name as the string passed to it:

```
ptPtr := Get pointer("tObject1 ") is the same as ptPtr := ->tObject
```

You can use concatenation to build the string passed to the function.

```
Get pointer ("tObject"+"1 ")
```

also creates a pointer to tObject1. This allows you to quickly build pointers to a large number of similarly named objects. For example, to reference tObject1 ... tObject10, the following would create an array of pointers each of which points to one of the tObjects:

```
For($i; 1; 10)
  ptObject{$i} := Get pointer("tObject" + String($i))
End for
```

To refer to tObject6, you may use ptObject{6}->.

To quickly assign values from an array to all the objects, you would use the following instead of having ten assignment lines:

```
For($i; 1; 10)
  ptObjectl{$i} := atMyText{$i}
End for
```

You can take this process one step further and combine the referencing and de-referencing into one step.

```
For($i; 1; 10)
  (Get pointer("tObject" + String($i)))-> = atMyText($i)
End for
```

About the Author

Michael P. Goldsmith is the president of Small System Solutions, 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. (If you are new to pointers you may want to read Mike's brief guide to pointers on page 72.)