## # of Invoices (Construction Hours):

Firstly, we have an overall control **=IF($C$3=FALSE, “-”.** This will return a dash, “-”, if the control cell is ‘False’ and will return the calculations if the control cell is ‘True’. This lets us turn off/on the worksheet at our command.

**(OFFSET(GFS!$A$1,1,0,$C$5,1)=B15)**

**$C$5 is calculated as COUNTA(GFS!A:A)-1**

We first use the OFFSET function to create an array of employee names. To properly create the array, we must find the size of the array. The width is hardcoded as “1”, but the number of rows is referenced as cell $C$5. This is calculated by counting all non-empty cells using the COUNTA function on the first column of GFS. The number is then subtracted by 1 to get rid of the header in the column.

We then logically test to see if these values in the array are equal to the specified cell (for the summary of the left, this is employee name and for the summary on the right, this is job type). This logical test will create a Boolean array of ‘1s’ and ‘0s’, with ‘1s’ representing a match to the specified cell.

**IF($E$3=FALSE,1,(OFFSET(GFS!$B$1,1,0,$C$5,1)>=$E$4)\*(OFFSET(GFS!$B$1,1,0,$C$5,1)<=$E$5))**

The next two uses of OFFSET, will create two Boolean arrays of 1s and 0s, sized using the same method as above, and multiplied together to create a combined array of 1s and 0s. If the date tested is above the start date specified, the first array will return a “1”, and if it is below the end date the second array will return a “1”. Therefore, the new array will only return a one if the date is between the two dates. (as 0\*0=0, 1\*0=0, 0\*1=0, 1\*1=1). These two offsets are controlled by an IF function that is dependent on the “Use Date Filter” field. This allows the user to apply or remove this date filter.

Finally, these two Boolean arrays that check for employee name/job type and dates will be multiplied together to create a final Boolean array. If all conditions are met, the field will return a ‘1’. The use of the sum function will then sum all ‘1s’ on the array, displaying total numbers of invoices per person. This is why it is important for this formula to be an array formula, or else the sum will not sum over the entire array.

**= (Boolean Array of Name/Job)\*(Bool array of min date)\*(Bool array of max date)**

## Hours

The “hours” array function is similar to the Number of Invoices formula, however an additional array is multiplied before the array is summed. The array of 0s and 1s is multiplied by an array containing the hours worked on each project. The same logic applies as in Number of Invoices, so if the name or date do not match, the final array created will return 0. If it does match, the final array will contain the number of hours worked. The array is then summed to get the total hours for each name or job type.

**= (Boolean Array of Name/Job)\*[(Hours)]\*(Bool array of min date)\*(Bool array of max date)**

## Travel

The travel function is the exact same as the hours formula, however the hours column in GFS is substituted with the Travel column. This creates a total sum of travel hours instead of regular hours.

**= (Boolean Array of Name/Job)\*[(Travel)]\*(Bool array of min date)\*(Bool array of max date)**

## Total Costs Excluding Markup

Total costs excluding markup also uses the boolean tests for name/job type and date used above. This array is multiplied by the array of hours plus the array of travel, essentially multiplying the boolean by the total number of hours worked per job. This is then multiplied by an array of rates. By multiplying rate by total number of hours, we return an array of total costs per job. By summing this array, we summarize the value into total cost per name or total cost per job type.

**= (Boolean Array of Name/Job)\*[(Hours) + (Travel)]\*(Rate)\*(Bool array of min date)\*(Bool array of max date)**

## Total costs Including Markup

The only difference in this array formula is that the product of travel and rate is also multiplied by the markup cell, given as a percentage increase, corresponding to that row. The total array formula is displayed below:

**= (Boolean Array of A1)\*[(Hours) + (Travel)]\*(Rate)\*(Markup)\*(Bool array of min date)\*(Bool array of max date)**

## Checks of the totals

The checks in row 10 test to see if the total sum in the row 9 is the same as the total sum in the GFS worksheet. It sums all the values in between the start and end dates in the GFS worksheet, and subtracts the total sum in row 9, which was calculated using a normal sum function in the current worksheet. This tests to make sure all data in GFS is accounted for in our model, and will catch some formula or input errors.

## Difference and Indirect Formula

The difference between the two worksheets is the INDIRECT function that is added in all the array formulas in the DASHBOARD WI worksheet, instead of hardcoding the sheet name “GFS” in every formula. The INDIRECT function is useful because it allows for dynamic use of references by taking a text or number input and creating a cell reference. In other words, it creates references in real time as the formula is evaluated.

In this example, the INDIRECT function is being used to reference the 'GFS' worksheet by using the string in the C4 cell. Say another worksheet was added to this document called 'TPY'. If you wanted to take the data from the worksheet 'TPY' the same way you previously did 'GFS', you would have to code everything over again. But by using the indirect function, you can softcode the reference to the worksheet to a single cell. Basically, you don’t need to change your model with new inputs.

This will be useful if you want to quickly compare data sets. For example, you may have one sheet of 2010 data and one sheet of 2009 data, and want to quickly compare. Alternatively, you may have data from multiple geographic locations. By using data tables, data from multiple sheets can be compared side by side with only one model created. This will make dashboarding our data more efficient and easier to read.

There are risks to this method. By adding one more calculation within each formula, our model becomes marginally slower computationally. This will become increasingly noticeable as the number of rows becomes larger.