**DATA PROCESSING PROCEDURES 05:**

CSPRO TABLES IN BATCH MODE

**Table of Contents**

[Introduction 3](#_Toc52605504)

[1. Tables in Batch 3](#_Toc52605505)

[1.1 Creating a Batch Application 3](#_Toc52605506)

[1.2 Batch Application Files 5](#_Toc52605507)

[2. Order of Operations in CSPro Batch Applications 8](#_Toc52605508)

[3. Table Declaration Syntax 11](#_Toc52605509)

[3.1 Overview 11](#_Toc52605510)

[3.2 Table Definition 11](#_Toc52605511)

[4. Using More than One Variable in Rows, Columns, and Layers 15](#_Toc52605512)

[5. The Working Storage Dictionary 17](#_Toc52605513)

[6. Post Processing 20](#_Toc52605514)

[6.1 Post Processing Entire Tables 20](#_Toc52605515)

[6.2 Post Processing Individual Cells 20](#_Toc52605516)

[6.3 Post Processing Sub-Tables 21](#_Toc52605517)

[6.4 Table Coordinates 23](#_Toc52605518)

[7. Post Processing Functions 26](#_Toc52605519)

[7.1 Functions TBLROW, TBLCOL, and TBLLAY 26](#_Toc52605520)

[7.2 Computing Percentages Using Functions 27](#_Toc52605521)

[7.3 TBLSUM Function 31](#_Toc52605522)

[8. Tabbing Multiple Times 37](#_Toc52605523)

[8.1 Post Processing 41](#_Toc52605524)

[8.2 Creating Tables with Layers 42](#_Toc52605525)

[9. Creating Tables with Nested Variables 45](#_Toc52605526)

[10. Weights 48](#_Toc52605527)

[10.1 Sampling Weights and Design Weights 48](#_Toc52605528)

[10.2 Normalization of Sampling Weights 48](#_Toc52605529)

[10.3 Main Sampling Weights in DHS Data 48](#_Toc52605530)

[10.4 Domestic Violence (DV) and HIV Sampling Weights 49](#_Toc52605531)

[10.5 Application of Sampling Weights in CSPro 49](#_Toc52605532)

# Introduction

CSPro can produce multi-dimensional tables that include totals, percentages, means, and medians, and can calculate rates based on results in other tables. There are two features of CSPro tabulations that set it apart from other software.

First, CSPro provides the programmer with tremendous flexibility in terms of when results are tabulated into a table. For example, one can tabulate data into a table multiple times in a single case. This allows the user to tabulate data into individual rows and/or columns at different times rather than tabulating the entire table once. As a consequence, CSPro can have multiple denominators in a single table. This flexibility simplifies the creation of complex tables.

Second, CSPro allows the user to manipulate tables as multi-dimensional arrays after all data have been processed. This allows the calculation of complex indicators such as fertility and mortality rates, as well as percentages, medians, and means.

**IMPORTANT:***To successfully compile and run CSPro tabulations in batch, CSPro requires that an environment variable* ***“*OLD\_TABS”** *be set as a user variable in WINDOWS. To do so, from the desktop, right click on “My Computer” in Windows 7 or 8; or right-click on “My PC” if you are using Windows 10. From the system menu choose “Advanced system setting” and a dialog box will open. Click on the button “Environment Variables”. This new window has 2 panels: one for User Variables and another for System variables. Then add a new User variable named* ***OLD*\_TABS***by selecting “NEW” and entering OLD\_TABS as the requested variable name and* “**Y**” *as the variable value.* ***NOTE:*** *When this variable is set to* “**Y**”*, the interactive CSPro tabulation tool will not function. When you want to use this utility then change* “**Y**” *to* **“N*”.*** *Additionally, if CSPro was open before the environment variable was set, CSPro should be closed and then reopened to enable batch tabulations.*

When a CSPro batch program that generates tables runs, it creates a .TBD file with the same name of the application that was used to create the tables. This is a binary file that contains all the information required to print a table. Information such as the values in the cells, the title, the variables in each dimension, the variable and value labels, the number of decimals to be printed, among others, is contained in the .TBD file. The values in the cells correspond to the results after the application passed the post-processing, if no postprocessing statements have been written, the values in the cells will be absolute numbers.

The purpose of the “TablesEditor” utility is to facilitate the exporting of a TBD file into Rich Text Format (RTF), which is a readable format across many programs and platforms as Microsoft Word. TablesEditor uses the .TBD file as input to produce tables that resemble the way that they are presented in the DHS final reports (camera-ready tables). Most of the modifications are done automatically while for other edits it is necessary to instruct the program either interactively or using the parameters file (.PRM).

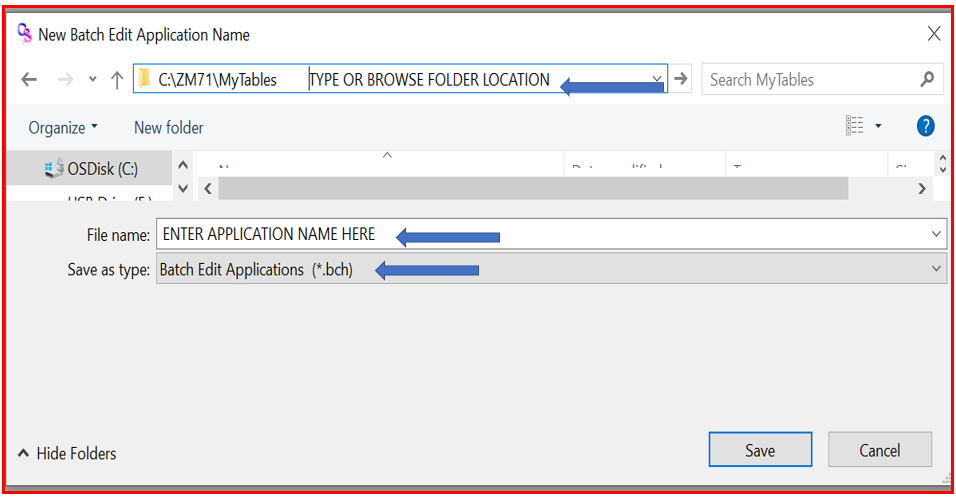
# 1. Tables in Batch

1.1 Creating a Batch Application

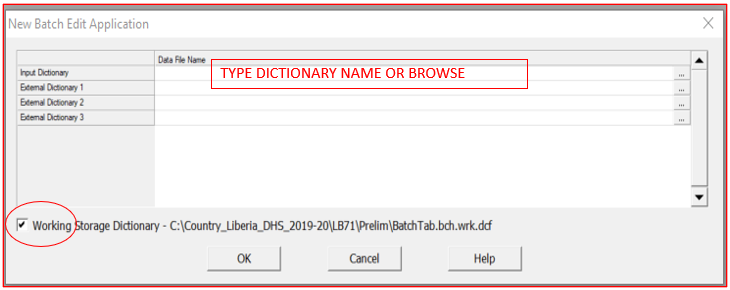
To generate tables with CSPro in batch, a batch application must be written, saved, and executed from the main menu. The first step in creating a tabulation application begins at CSPro’s main menu. There are two ways to create a new batch application:

* After launching CSPro select the ‘**Create a new application**’ radio button from the dialog box, or. 
* If CSPro is already open, from the **File** menu, select **New.**

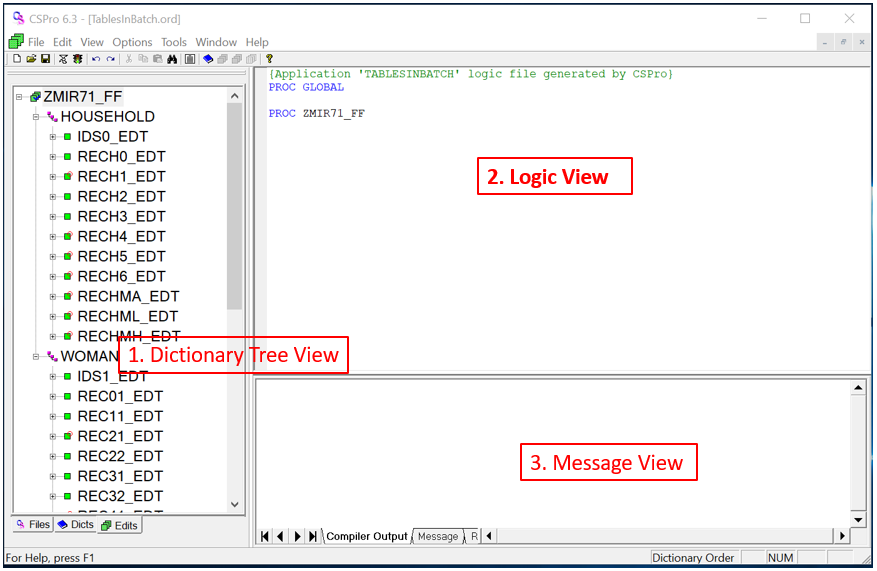
A dialog box will appear,select **Batch Edit Application,** and press **OK.** Another dialog box will open ‘**New Batch Edit Application Name**’. Enter the name of the application file in the *‘File name’* box. The *‘Save as type’* box indicates that this will be a new batch edit application with a **\*.bch** extension. Before pressing the **Save** bottom, make sure you set the folder location where the applications will be placed by typing the path or browsing. Choose e.g. “TablesInBatch” as the file name and select ‘**Save**’.



By pressing “Save”, another screen will display. The input dictionary associated with the dataset to be used with this application must be entered in the ‘**New Batch Edit Application**’ window. Type the dictionary name or click  to browse for the dictionary name. Check the box to use the default working dictionary provided by CSPro. When this dictionary is never used, it can be deleted later. No external dictionaries will be used, so press **OK**.



By pressing OK, CSPro will create a batch application and will display the following screen layout to start writing logics.



The screen is split into three working areas: The **Dictionary Tree View** to the left, the **Logic View** to the upper-right, and the **Message View** to the bottom-right. CSPRO will provide two lines of codes as part of the Logic View panel. These two lines will always be in the application as follow and *cannot be deleted*.

|  |
| --- |
| {Application ' ***TableInBatch*** ' logic file generated by CSPro}  PROC GLOBAL  PROC ZMIR71\_FF |

*The size of the tree working areas can be modified; just place the mouse over one of the separating bars,*

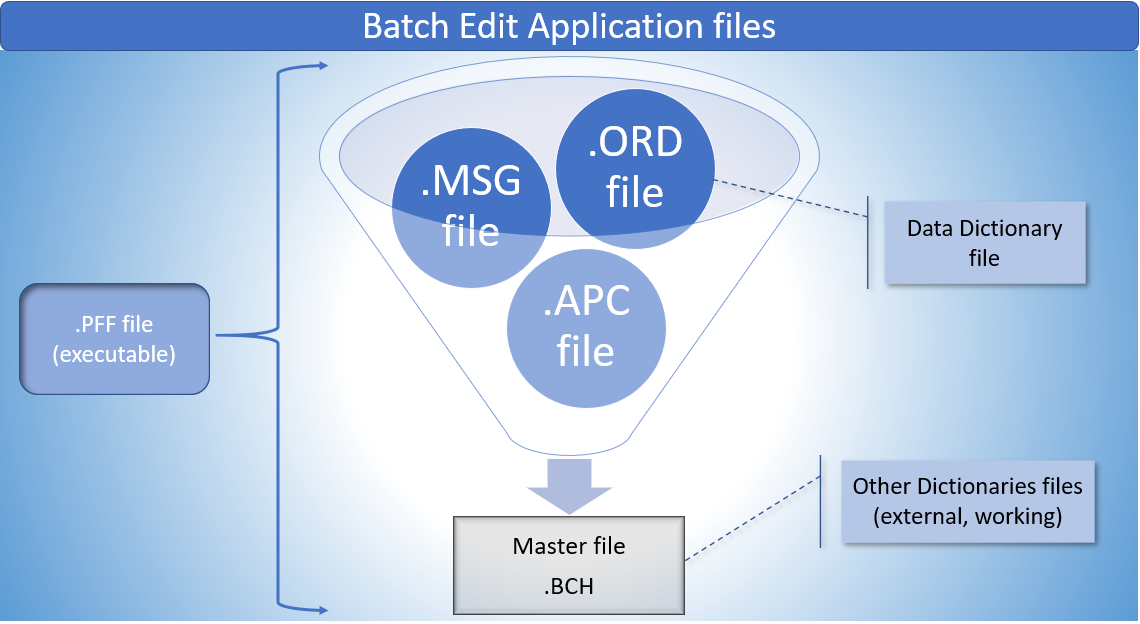
*grab it and drag to resize.*

1.2 Batch Application Files

When a new batch edit application is created, CSPro generates four basic files as part of the application. All the files will have the same name as the application’s name but a different extension: .**BCH**, .**ORD**, **.MSG**, **APC** *(before CSPro 6.0, the extension .APP was used instead of .APC)*. If any of these four files is deleted by error, CSPro will not be able to open the application, instead, an error will be displayed.

There is another file with the extension **.PFF** which is generated after the batch application is executed. If the option “create a working dictionary is selected, CSPro will generate an additional file with extension **.DCF**. This is a working storage dictionary generally use to define working variables.

All files can be viewed using a text editor like notepad or CSPro Text Viewer, however, it is not recommended to make changes to any file outside of the CSPro environment.



|  |  |
| --- | --- |
|  |  |
| **Batch file (.BCH)** | This is the master file where all the other files contained in the application are specified along with others information. When a batch application is opened, saved, or closed, all other files associated with the application are also opened, saved or closed, respectively. |
| **Edit Order File (.ORD)** | This file is to specifies the order in which logic in the application is executed. It is a text file that contains information of the sequence of fields associate with the data dictionary. All data dictionary item will be listed in this file. |
| **Message file (.MGF)** | This is a text-format file to store messages to be displayed. Only one message per line is allowed. A message consists of a message number followed by text. The text can be 240 characters long and may contains parameters. Messages can be displayed using the function “**errmsg**”. |
| **Logic file (.APC or .APP)** | This file contains all the logic, written in CSPro language, that control the application. There is only one logic file associated with each batch application. Prior to CSPro 6.0, logic files had the extension (.APP). However, CSPro can still read applications that have logic files with the **.app** extension. |
| **Working storage dictionary (.WRK.DCF)** | This is an additional dictionary that can be created automatically by CSPro. It is similar to any other dictionary, but no data associated with it. This dictionary appears just below the main dictionary in the dictionary tree area. |
| **Program Information File (.PFF)** | This file is used to run CSPro applications like entry, sort, batch, etc. This application stores the names of the application, the data file(s) and any other runtime parameters .PFF files can be used as a command line parameter. |
| **Data Dictionary files (.DCF)** | This file represents the primary data file that will be read and/or written. Each .ORD file contains one Data Dictionary file (.DCF). CSPro requires a data dictionary at the time when the batch application is created. |

|  |
| --- |
| **Exercise 1** |
| * *Make sure that you have copied the directory* ***ZM71*** *(Zambia DHS7) with the following subdirectories to your hard drive*   + *\DICTS*   + *\FINDATA*   + *\TABLES*   + *\LIBRARY*   + *\TablesEditor*   + *\Frequencies* * *The dictionaries “****ZMIR71.DCF****” (households/women individual recode dictionary) and* ***ZMMR71.DCF*** *(male individual recode dictionary) will be located in the DICTS folder* * *and the data files* ***ZMIR71.DAT*** *(households and women data) and* ***ZMMR71.DAT*** *(male data) will be located in* ***\FINDATA****.* * *Following the above instructions, start your new batch application.* * *The name of the application should be “****MyBatchTables****”* * *Create a new folder named* ***\MyTables*** *where all your exercises will be stored* * *The application should be saved in the* ***\MyTables*** *folder* |

# 2. Order of Operations in CSPro Batch Applications

The structure of a batch application is the same as for a data entry application, and it is summarized below. Batch applications make more extensive use of dictionary level procedures (and rarely use variable procedures) than data entry applications. In batch applications the timing of statements regarding specific variables is less crucial, and statements can be placed in dictionary level procedures where they are easily read.

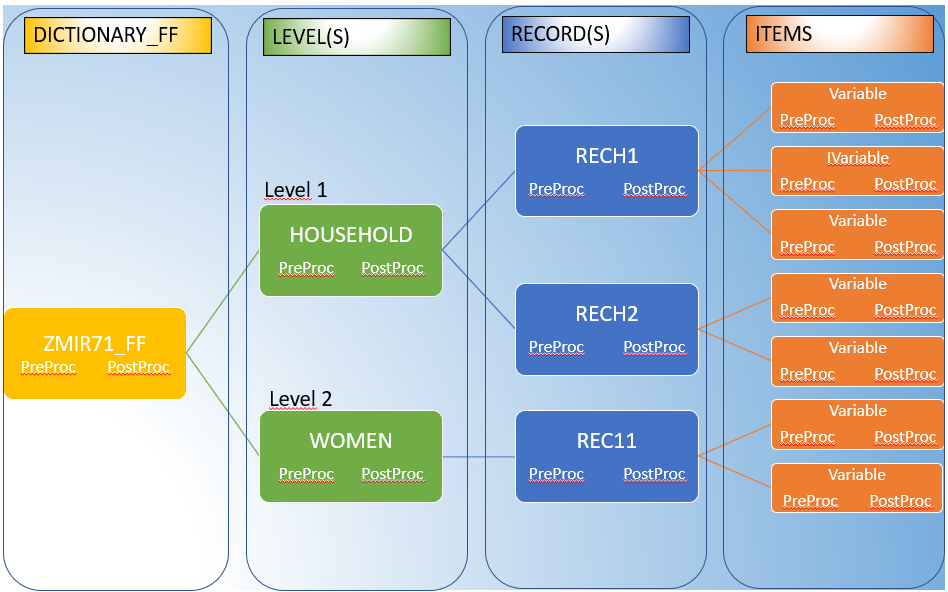
Different from other statistical packages, CSPro executes each line of logic one case at the time but its execution will follow a specific order depending on the CSPro dictionary’s elements and procedures. Each dictionary element has two types of procedures: preproc and postproc. If no procedure is specified, CSPro will assume **postproc** as default. The logic should be written in the CSPro language.

As mentioned above in point 1.2.1, when a batch application is created, CSPro will write two lines of code: **PROC GLOBAL** and **PROC DICTIONARY\_FF**. These are the two main entities that will always be present and cannot be deleted.

PROC GLOBAL section is used for declaration and definition of statements that can be used at any time by any procedure (globally). What can be declared in this section?

|  |  |
| --- | --- |
| **Declarations** | **Explanation and Rules** |
| **Mode of operation:** | CSPro compiler has two modes of operation: **Explicit** or **Implicit.** The default compiler mode is **Explicit**. The default setting can be overwrite using the “set” command as “Set Implicit”. |
| **Explicit Mode**  **(Set Explicit)** | When set explicit mode is used, all temporary variables that are not defined in your dictionary(ies) MUST be declared in a numeric or alpha statement; otherwise, the variable will be flagged as error. |
| **Implicit Mode**  **(Set Implicit)** | When set implicit mode is activated, temporary variables can be declared “on the fly”, anywhere in the application. Using this mode can be faster to write codes but also be risky if the variable name is misspelled. Any variable misspelled will be declared by CSPro as a new variable with initial value of “0”. Therefore, any logic statement using this new variable will lead to a wrong result. |
| **Working variables**  **(alpha or numeric)** | There are two types of variable: **numeric** or **alphanumeric**. Variable names can be 32 characters long, MUST begin with a letter and MUST contain only letters, numbers, or the underscore(‘**\_**’) character.  Alpha statement or string statement are used to declare a string-variables, and numeric statement for numeric variables. |
| **Arrays** | Arrays can be numeric or alphanumeric and up to three dimensions. Array names follow the same rules than variables names (see above). Only one variable can be defined in each array statement. The initial value of an alpha array is blank and zero for a numeric array. |
| **Functions**  **(user-defined)** | Functions are piece of code defined in Proc Global that can be used anywhere in an application. Instead of writing the same codes multiple times, a function can be declared and every time the code is needed, the function can be called. One function commonly used in the DHS batch applications is called **“CMCode(month, year)”**, this function will convert dates to months since 1900 (Century Month Code).  Function returns a single value, either numeric or alpha and functions can be called within a function. |

PROC DICTIONARY**\_**FF section can be used for executable and assignment statements. Those statements can be written under a **preproc** or **postproc** procedures. In a batch tables application, the dictionary **preproc** is typically used for assigning values to working variables, while the **postproc** is used to carry out the post-processing of tables (see section 5.1 below). Each element of a dictionary has pre and post processing procedures so the logic can be written before or after an event. A dictionary has the following elements: **levels**, **records**, **forms, rosters**, and **fields**. Forms and rosters are elements used mainly in data entry applications.

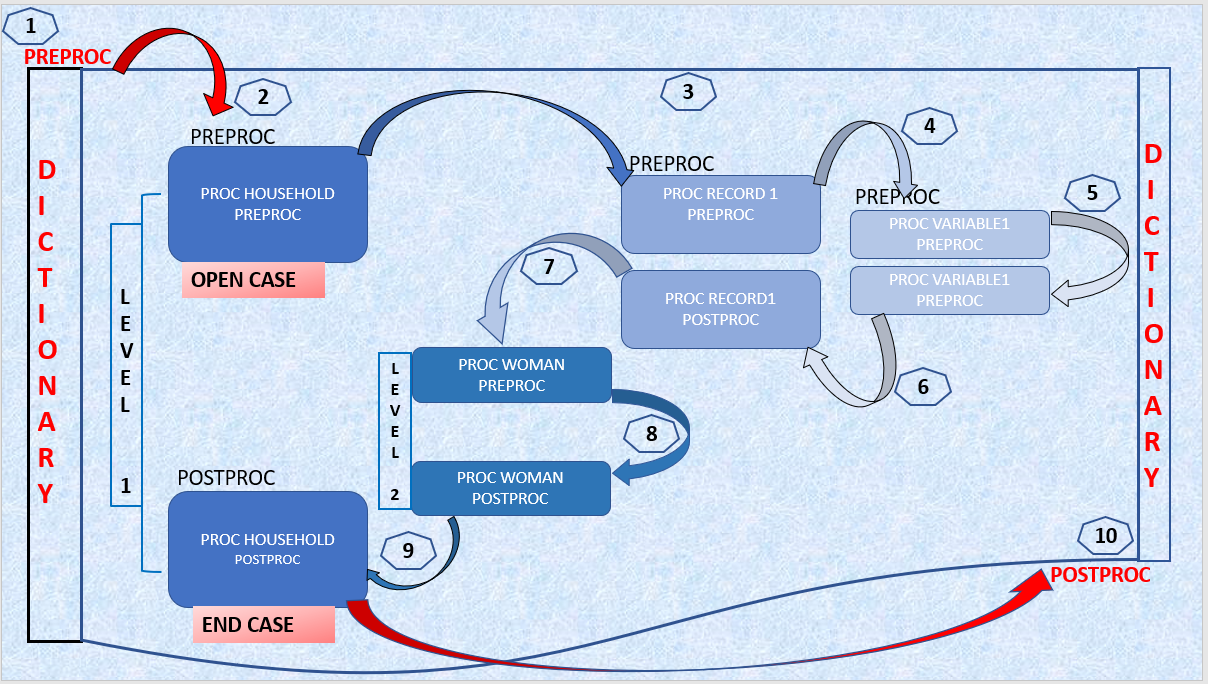


When a batch application is run, the **PROC GLOBAL** is executed first. The dictionary level procedures are executed next in the following order. The **PreProc of the dictionary** is the first to be executed and a*ny logic written in this node will execute only once for a data file*. It works similarly to **PROC GLOBAL** where any variable assignment can be used globally. Any global variables can be initialized in the dictionary **PreProc**. The procedures for the remaining dictionary levels are then executed based on the following principle: each level preproc is executed before the preproc and postproc of the following level, and each level postproc is executed after the preproc and postproc of the following level.

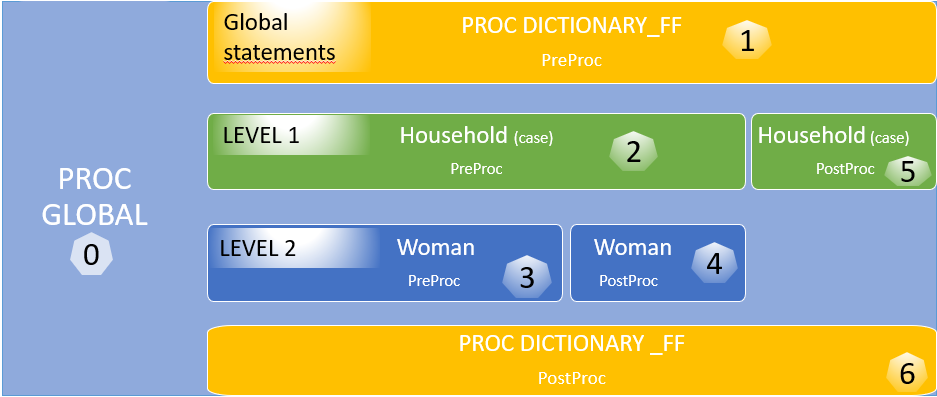
Logic written on **level-1** will only execute once for a questionnaire/case. Logic written for level 2 or 3 will execute for each record contained in that level.

The last procedure to be executed is the **Dictionary\_ff postproc.** It is in that node where all statistics (percentages, mean, median, rate, etc.) for a specific DHS’s table are calculated.

**Order of executing procedures for tables batch application**

****

For TablesInBatch applications, the order of operation is much simpler than for a DataEntry applications. Mainly the Dictionary preproc/postproc and levels preproc/postproc are used to write logics. The graph below shows the path while executing procedures within levels.



# 3. Table Declaration Syntax

3.1 Overview

The programming for CSPro batch tables has the following three components:

1. **Definition** of the table in the PROC GLOBAL, using the crosstab command and subcommands
2. **Tallying** of values into the table’s cells using the **xtab** function in procs in level 1 or lower
3. **Post-processing** of tallied absolute values in the tables cells in the dictionary **postproc** to calculate percentages, medians, means, rates, or other indicators. Note that this last step may be omitted if only tables of integers or floating-point numbers are desired without any additional calculations.

3.2 Table Definition

Tables in batch are generated by using the CROSSTAB command and subcommands in combination with the **XTAB** function. The structure of a table is defined using the CROSSTAB command and subcommands in the PROC GLOBAL procedure and each cell of the table will be tallied using the **XTAB** function (in the relevant ‘Level’ procedure).

The CROSSTAB command defines the structure of the table by specifying the row, column, or layer variables that it contains.

The syntax of the CROSSTAB command is as follows:

|  |
| --- |
| **CROSSTAB** [ float() | sint | lint ] Table\_name row variable+list, [column variable+list]  [layer variable+list]  Exclude[ (*rowzero, colzero, percents, totals, specval*) ]  Include[ (*rowzero, colzero, percents, totals, specval*) ]  Title( *“title 1\_line 1”, “title 2\_line 2, “title 3\_line 3”, “….”*)  Stub*“title 1\_line 1”, “title 2\_line 2, “title 3\_line 3”*,) |

Subcommands that are required in a table definition:

|  |  |
| --- | --- |
| **Required** | **Description** |
| **SINT (Short integer)**  **LINT (Long integer)** | The choice between SINT and LINT should be based on the maximum cell value and the amount of memory available in the computer. Each cell of a SINT table requires 2 bytes, whereas each cell of a LINT table requires 4 bytes. SINT will generally provide sufficient accuracy for integer tables. The maximum value in any cell in a SINT table is ±32,767. In a LINT table, this maximum is ±2,147,483,647. Note that SINT and LINT should only be used for tables showing integer values and not percentage or other calculated values such as means, medians or rates. |
| **FLOAT( )**  **(floating point)** | The contents of each cell in a FLOAT() Table are represented as double precision floating point numbers, i.e. numbers with decimal places. In a FLOAT Table, (n) is the number of digits to include to the right of the decimal point when printing the table. **When using weighted data, always define the table as FLOAT()**. To automatically round all cells in a FLOAT table to omit decimals when printing the table, use FLOAT(0). Each cell of a FLOAT table requires 8 bytes. |
|  | *Only one of these three (SINT, LINT, FLOAT() ) can be included in any one CROSSTAB Command. If none of them is specified, FLOAT is assumed.* |
| **TABLE NAME** | The name of the table is a required parameter in the **CROSSTAB** command. **TABLE\_NAME** can be any combination of up to 8 characters, starting with a letter, and may include letters, numbers, and the underscore character. The table name is used later in the application for all references to the table (including as a parameter in the **XTAB** function). |
| **DIMENSIONS:**   * **ROWS** * **COLUMNS** * **LAYERS** | Tables are composed of rows, columns, and optionally, layers dimensions. Each row, column, and layer consist of variables from any dictionary used in the application including working storage variables (as discussed later in this chapter). Rows, columns, and layers can each consist of one or more variables.  Each dimension, the row variable list, column variable list and layer variable list may optionally be separated from each other by commas for clarity or by blank space. |
| **“ + ”** | The plus sign “+” is used to separate variables listed within each dimension, ROW, COLUMN or LAYER (if more than one variable). |

Subcommands that are optional in a table definition:

|  |  |
| --- | --- |
| **Optional** | **Description** |
| **TITLE(“….”)** | The **TITLE** is an optional parameter, to be displayed as the table’s title in the output. Each line of the title is enclosed in double quotes. Lines are separated from other lines with a comma, where each comma signifies the beginning of a new line in the output title (not in the syntax). The entire title is enclosed in parentheses. |
| **STUB** | The **STUB** is an optional parameter and has the same structure as the TITLE parameter. Each line of the STUB is displayed in the upper left corner of the table, at the intersection of the row and column value labels |
| **EXCLUDE**  **INCLUDE** | The **EXCLUDE/INCLUDE** subcommand are used to list features/statistics that should be omitted or included from the table. These features/statistics are summarized in the following table. By default, only **Missing** and **Default** special values are included in a table.  **List of parameters or special values that are included/excluded by default:**   |  |  |  |  | | --- | --- | --- | --- | | **Parameters** | **Description** | **Default** | | | **SPECVAL** | MISSING, NOTAPPL, DEFAULT |  | | | MISSING | Missing values | | Included | | NOTAPPL | Not applicable values (blank) | | Excluded | | DEFAULT | Default values (\*’s, special characters) | | Included | |  |  | |  | | **ROWZERO** | Rows are all zeros | | Included | | **COLZERO** | Columns are all zeros | | Included | | **LAYZERO** | Layers are all zeros | | Included | |  |  | |  | | **TOTALS** | ROWTOT, COLTOT, and LAYTOT | |  | | ROWTOT | Row totals | | Included | | COLTOT | Column totals | | Included | | LAYTOT | Layer totals | | Included | |  |  | |  | | **PERCENTS** | ROWPCT, COLPCT, LAYPCT, and TOTPCT | |  | | ROWPCT | Percentages across rows | | Excluded | | COLPCT | Percentages down columns | | Excluded | | LAYPCT | Percentages among layers | | Excluded | | TOTPCT | Percentages based on table totals | | Excluded | | **FREQ** | Actual cell contents | | Included | |  |  |  | | |

In most DHS table declarations “rowzero”, “colzero”, “specval”, “totals” and “percents” are included in the “**EXCLUDE”** parameter. Note that “specval” includes “missing”, “NotAppl” and “default”; therefore, if a standard recode variables is used in the table definition then the “missing values” will not be included in the table. This is why nearly all variables used as DHS table definitions are working variables, where the missing cases are recoded to a value (usually 9 or 99). Likewise, “totals” and “percents” are excluded, forcing these rows and columns to be explicitly programmed or computed in the application; thus increasing readability and standardization across applications.

The individual components of the **CROSSTAB** command can be written on one line or separate lines. As with all CSPro commands and functions, the **CROSSTAB** command is not case sensitive.

The **XTAB** function is used to tally data into the table. The **XTAB** function’s syntax is as follows:

|  |
| --- |
| **Xtab** ( TableName, [arithmetic\_expression] ); |

Every time the **XTAB** function is executed, the cell (or cells) that represents the values of the row, column, and/or layer variables are incremented by a value equal to the arithmetic expression, or 1 if no arithmetic expression is used. This arithmetic expression is typically used to weight the tally by the case weight. See section 10 “Weights” for more information on when and how to use sample weights. The table name parameter specifies to which table tallying should be performed.

|  |
| --- |
| **Exercise #2** |
| * *Enter as shown below the example for a* ***CROSSTAB*** *command under* ***PROC GLOBAL*** * *Enter the “****Xtab( example1)****” command on the* ***PROC WOMAN*** *level because this table will tally individual interview (see dummy table below)* * *Run the application using the data file* ***ZMIR71.DAT*** *under \FINDATA folder* * *Use TablesEditor to export the .TBD file to RTF* * *A similar table to the one below should be produced*   **{Application 'MYBATCHTABLES' logic file generated by CSPro}**  **PROC GLOBAL**  **CROSSTAB float(1) example1 V015, V025**  **exclude(rowzero,colzero,specval,totals,percents)**  **title("Example 1.Results of individual interviews",**  **" ",**  **"Percent distribution of results of individual",**  **"interviews by urban/rural residence",**  **"Zambia DHS 2018-19");**    **PROC ZMIR71\_FF**  **PROC WOMAN**  **Postproc**  **xtab( example1 );** |

# 4. Using More than One Variable in Rows, Columns, and Layers

CSPro tables are matrices of one, two or three dimensions. The first dimension consists of rows, the second dimension consists of columns, and the third consists of layers. Each dimension contains one or more variables. If a dimension contains more than one variable, all the variables must be separated from each other by "**+**" or "**\***". If the “+” is used, the Variables will appear adjacent to each other in the table. If the “\*” is used, the variables are “nested”. Nesting is not permitted in layers. In rows and columns, combinations of “+” and “\*” can be used, and variables can be grouped using parentheses. See some examples:

1. The examples used in this chapter so far have included only one variable in the rows and columns. Focusing on the columns, a table with only one column variable, such as V025, would have the following column headings:

|  |  |
| --- | --- |
| Type of Residence **(V025)** | |
| Urban | Rural |

1. If a second variable, V213 (“Currently pregnant”), was added to the columns with the “+” operator, the list of column variables would be **V025+V213**, and the table columns would appear as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Residence**(V025)** | | Currently pregnant**(V213)** | |
| Urban | Rural | Yes | No |

1. If the “ \* ” operator was used, V213 would be “nested”, the columns of the table would be defined as **V025\*V213**, and the column headings would appear as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Residence**(V025)** | | | |
| Urban | | Rural | |
| Currently pregnant**(V213)** | | Currently pregnant**(V213)** | |
| Yes | No | Yes | No |

1. Reversing the order of the variables in the column definition to V213\*V025 would generate a table with the following columns:

|  |  |  |  |
| --- | --- | --- | --- |
| Currently pregnant**(V213)** | | | |
| Yes | | No | |
| Type of Residence**(V025)** | | Type of Residence**(V025)** | |
| Urban | Rural | Urban | Rural |

1. The “+” and “\*” operators can be combined in the same table. For example, if the columns are defined as V025\*V213+V015, a table with the following column headings would be generated:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Type of Residence**(V025)** | | | | Result of Interview **(V015)** | | | | |
| Urban | | Rural | |
| Currently pregnant**(V213)** | | Currently pregnant**(V213)** | | Complete | Not at home | Refused | Partly  Com-pleted | Incapa-  citated |
| Yes | No | Yes | No |

1. It is not possible to nest variables inside a nesting. Only one variable is allowed to the left of the “\*” operator, but it is possible to use a “+” operator to the right of it. Valid and invalid examples are shown below:

|  |  |
| --- | --- |
| **Valid** | **Not Valid** |
| VarA \* ( VarB + VarC )  VarA \* VarB + VarC  VarA + VarB \* VarC + VarD | (VarA + VarB ) \* VarC  VarA \* VarB \* VarC  (VarA + VarB ) \* (VarC + VarD) |

1. If all of these variables have two values, then the structure of the table columns for the first valid combination of variables (VarA \* ( VarB + VarC ) ) will appear as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| VarA | | | | | | | |
| VarA1 | | | | VarA2 | | | |
| VarB | | VarC | | VarB | | VarC | |
| VarB1 | VarB2 | VarC1 | VarC2 | VarB1 | VarB2 | VarC1 | VarC2 |

*All the examples shown above relating to columns can be used in the row dimension of a table. Table’s layers, however, cannot be nested and can only use “+”.*

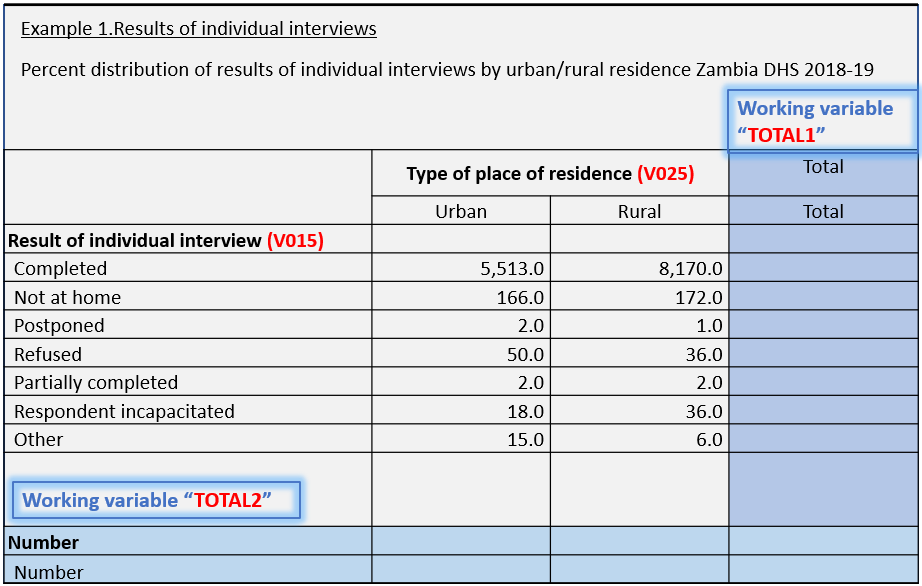
Group variables or variables in multiple sections can be used in tabulations only if a specific occurrence is specified, by using parentheses ().

# 5. The Working Storage Dictionary

It is often necessary when constructing tables to create new, customized variables for use in the table that differ from those used in the actual dataset being analyzed. In CSPro, this is done by creating variables in a working storage dictionary associated with the batch application. Once defined in the working storage dictionary, variables can then be defined as row, column, or layer variables in the crosstab command.

Working storage dictionaries are defined as external dictionaries. They are similar to any other dictionary but with the difference that they are not associated with a data file. That is, the values of variables in working storage dictionaries exist only in memory during the running of the batch application. As explained in point 1.2, it’s recommended while creating a new batch application to check the box “working storage” to use the default dictionary provided by CSPro (if this dictionary is never used, it can be deleted later).

Using the table in exercise #2, that shows the results of questionnaires for respondents in urban and rural areas. Another way to present this table would be by adding a column next to urban/rural to accumulate the total; and by adding a row under the result of the interview to accumulate the total.



CSPro will only tabulate variables that have been defined in a dictionary. Since variables TOTAL1 and TOTAL2 are not part of any dictionary, these two variables must be added to a working storage. The working storage dictionary has the same structure as an input dictionary. The working dictionary can be the one produced while defining the application (**MyBatchTables.bch.wrk.dcf**) or any other ***one level*** dictionary attached to the application by clicking on  and then choose “FILE”, “Add Files…”. Do not forget to set the type of dictionary as “Working” by doing right click on the dictionary, click on “type of dictionary” and select the radio button for “Working”. If “Working” is not selected, CSPro will assume that a data file is associated to this dictionary.

Adding new variables to the working storage is the same as adding new variables to any CSPro dictionary. These two new variables should be added to the working storage using the information below: variable name, labels, and length.

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Variable labels** | **Value labels** | **Length** |
| **-TOTAL1** | Total or **“.”** (\*) | 1 = Total | 1 digit |
| **-TOTAL2** | Number or **“.”** (\*) | 1 = Number | 1 digit |

(\*) *A full variable label can be entered if desired; since the variable and value label are the same, by entering a dot* ***“.”*** *as a variable label will cause the label not to appear in the table after it is exported by the tables editor.*

The location will be computed by CSPro and should not be changed. The length is the number of digits that the variable will require.

|  |
| --- |
| **Exercise #3** |
| * *Using the application “****MyBatchTables.bch****”, go to the working storage dictionary and add the two working variables that will be used to tabulate the total number of cases for urban/rural and interview result.* * *The working dictionary can be found at the end of the main dictionary tree; at the left side panel.* * *Since the main dictionary has many records, you must scroll down to be able to see the working storage* * *To access the dictionary, do not forget to click on the dictionary tab that can be found at the button of the dictionary tree.* |

Once the working variables have been added to the working storage dictionary, there are three steps needed to produce this table. As explain in point 4, the plus sign “+” is used by CSPro to add variables in any dimension of the table (row, column, and/or layer).

**Step 1**: Add the working variables to the row dimension and to the column dimension

|  |
| --- |
| **PROC GLOBAL**  **CROSSTAB** **float(1)** **example1** **V015+TOTAL2, V025+TOTAL1**  **exclude(rowzero,colzero,specval,totals,percents)**  **title(**"Example 1.Results of individual interviews",  " ",  "Percent distribution of results of individual",  "interviews by urban/rural residence",  "Zambia DHS 2018-19"**)**; |

**Step 2**: Initialize the working variables

|  |
| --- |
| **PROC ZMIR71\_FF**  **Preproc**  **TOTAL1 = 1;**  **TOTAL2 = 1;** |

The value of TOTAL1 and TOTAL2 should be set to “1” at the beginning of processing. The purpose of these two variables is to accumulate the number of cases by adding one case at the time. The appropriate place to initialize variables is at the **PreProc** of the dictionary level **“PROC DICTIONARY\_FF”.** Any logic written in this node will execute first and only once for a datafile. That is why we call it “initialization”. Once the value is been set to one, every time that **XTAB** function is executed, this will insert data into the “Number” of women row; and to the “Total” of urban/rural column. Keep in mind that the variables should be initialized based on the value-set we defined in the working storage. For this particular case, the variables were initialized equal to “1” because this is the value-set declared in the working storage; if the user prefers the value set = 2 or any other number from 1 thru 9, then the variables should be set equal to that value.

**Step 3**: Add the tally command “Xtab”, compile and run the application

|  |
| --- |
| **PROC WOMAN**  Postproc  xtab( **example1** ); |

**Step 4**: Check the output table using TablesEditor

|  |  |  |  |
| --- | --- | --- | --- |
| Example 1.Results of individual interviews | | | |
| Percent distribution of results of individual interviews by urban/rural residence Zambia DHS 2018-19 | | | |
|  | Type of place of residence | |  |
|  | Urban | Rural | Total |
|  |  |  |  |
| **Result of individual interview** |  |  |  |
| Completed | 5,513.0 | 8,170.0 | 13,683.0 |
| Not at home | 166.0 | 172.0 | 338.0 |
| Postponed | 2.0 | 1.0 | 3.0 |
| Refused | 50.0 | 36.0 | 86.0 |
| Partially completed | 2.0 | 2.0 | 4.0 |
| Respondent incapacitated | 18.0 | 36.0 | 54.0 |
| Other | 15.0 | 6.0 | 21.0 |
|  |  |  |  |
| Number | 5,766.0 | 8,423.0 | 14,189.0 |

*The new column and the new row added to the table are highlighted in gray.*

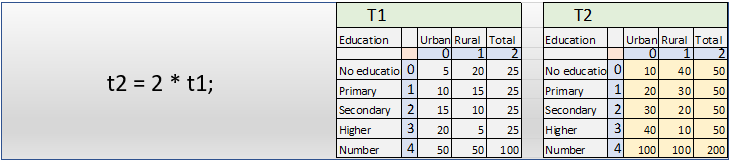
|  |
| --- |
| **Exercise #4** |
| * *Using the application “****MyBatchTables.bch,****”, go to the logic and add the two variables TOTAL1/TOTAL2 in the table declaration* * *Initialized the working variables* * *Run the application* * *Use tables-editor to export the table to RTF* * *Verified that the extra row and column are present* * *Table should look like the table above* |

# 6. Post Processing

Post processing is a powerful aspect of CSPro tabulations. It allows the user to manipulate the contents of a table after all data have been inserted into it. It allows calculations for individual cells, and entire rows, columns, and layers. Rows, columns, and layers can be added, subtracted, divided, and/or multiplied by rows, columns, and layers in the same table or in other tables. These operations can also be performed on entire tables. During post processing, each table can be treated as a 2 or 3 dimensional matrix, or array of values. Post processing of tabulations is done in the **PostProc** of the dictionary level (for Zambia this is ZMIR71\_FF).

6.1 Post Processing Entire Tables

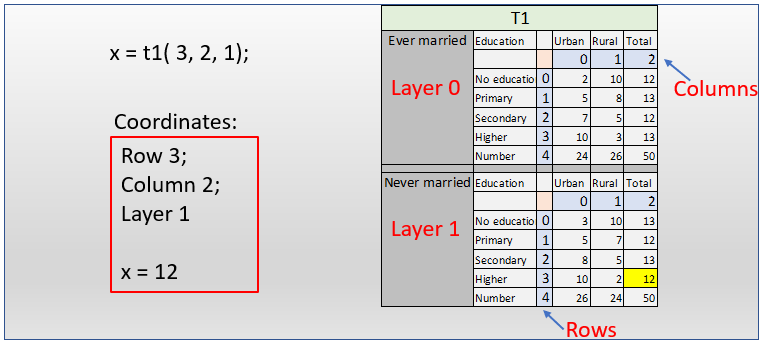
The most basic form of post processing makes use of simple arithmetic expressions. For example, if t1 and t2 are both tables, and t1 contains data, each cell of t2 can be assigned a value that is twice the corresponding value in the cells in t1: t2 = 2 \* t1;



In order to use this facility, both tables must have exactly the same structure, in terms of numbers of rows, columns, and layers.

6.2 Post Processing Individual Cells

More complex post processing involves the manipulation of specific table elements (rows, columns, and layers). References to specific rows, columns, or layers use parentheses, with row, column, and layer numbers separated by commas. The following example sets the value of x to the contents of the cell in row 3, column 2, layer 1 of table t1; the result is highlighted in yellow color in the table “T1” below: x = t1( 3, 2, 1);



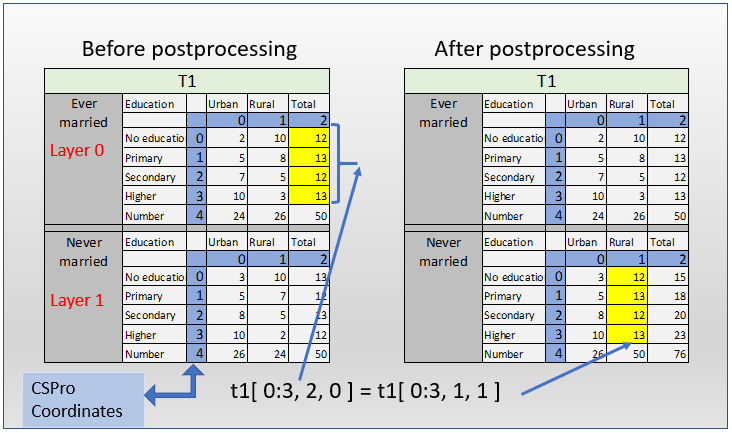
Individual cell contents in tables can be set the same way. The following example sets the value of the contents of the cell in row 0, column 1, layer 2 of table t1 equal to x:

**t1( 0, 1, 2 ) = x;**

6.3 Post Processing Sub-Tables

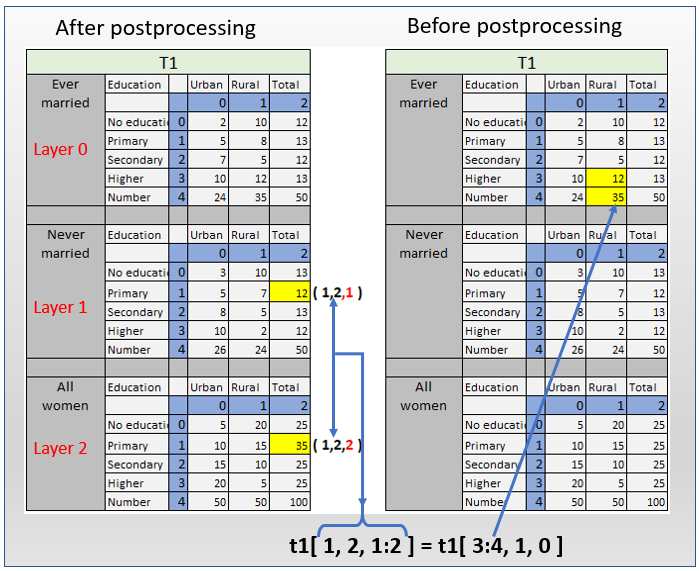
The most powerful feature of the post-processing is CSPro’s ability to post process selected parts of a table, or sub-tables. The syntax for referring to sub-tables is similar to the syntax used to refer to individual cells but uses brackets instead of parentheses. Additionally, sub-table processing uses ranges to specify the sub-table of interest. For example, the following statement copies [0:3,2,0] the values from row 0 – 3 of column 2, layer 0 into the rows 0-3 of column 1, layer 1 of table t1:

t1[ 0:3, 2, 0 ] = t1[ 0:3, 1, 1 ];



Note that CSPro numbers the rows, the columns and the layers of tables from 0, thus the first three rows are rows 0, 1, and 2. Also note that the sub-table on the left of the equal sign must have the same size as the sub-table on the right of the equal sign. In the example below, the values in rows 3 and 4 of column 1, layer 0 of t1 are copied into row 1, column 2, layers 1 and 2 respectively:

**t1[ 1, 2, 1:2 ] = t1[ 3:4, 1, 0 ]**

****

The following example is invalid as the range on the left-hand side of the equals sign is greater than the range on the right-hand side of the equals sign:

**t1[ 0:4, 2, 3 ] = t1[ 0:2, 1, 3 ]**

The next example is also invalid as CSPro does not allow the mixing of sub-table expressions with references to individual cells:

**t1[ 0:2, 1, 3 ] = t1( 1, 2, 3)**

In the last example, there was an attempt to assign a single cell value to several cells in a table. This can be done, but requires two statements, as follows:

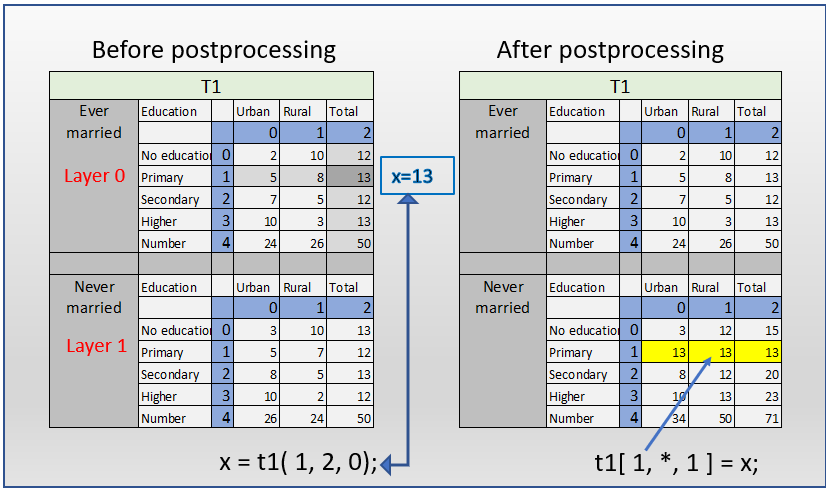
**x = t1( 1, 2, 0);**

**t1[ 0:3, 1, 1 ] = x;**

Tabulation post processing also allows the use of the asterisk as a wild code, meaning **“all”**. For example, the following statement sets all cells in row 1 and layer 1 of table t1 to **x**:

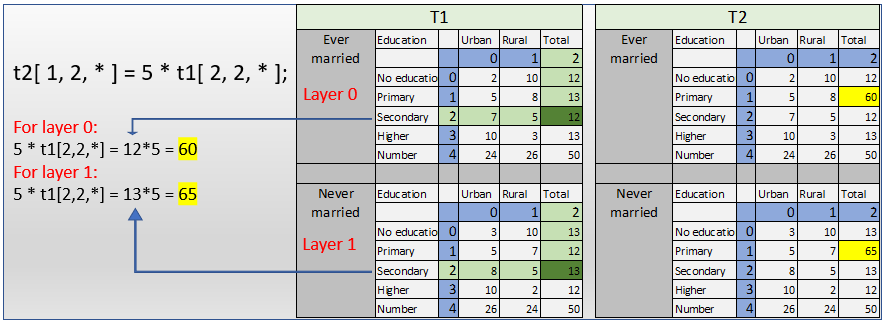
**x = t1( 1, 2, 0);**

**t1[ 1, \*, 1 ] = x;**



This same syntax can be used to set values in multiple rows, columns, or layers using cells in more than one table. The following statement sets the value at the intersection of row 1 and column 2 for each layer of table t1 equal to five times the value at the intersection of row 2 and column 3 for each layer of table t2:

**t1[ 1, 2, \* ] = 5 \* t2[ 2, 3, \* ];**



When using this syntax, the “source” dimension (row, column, or layer) must be equal in size to the “target” dimension. For example, you cannot set a 3-value column (the “target”) equal to a 4-value row (the “source”).

6.4 Table Coordinates

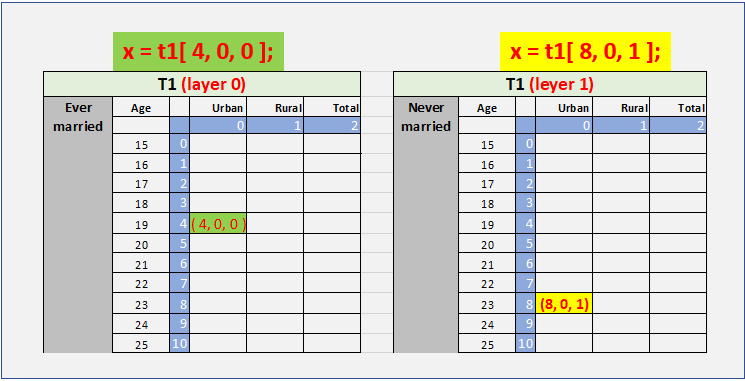
At this point it is important to point out that the indexes into the tables are not the same as the values of the variables that make up the rows, columns, and layers of the table. CSPro translates the values of the variables (valueset) into table coordinates because tables can have several variables on their rows, columns, or layers, and in order to minimize the allocation of memory needed for the table.

For example, a variable for the age of a woman might have a range of values specified between 15 and 49. When this variable is used as a row, column or layer variable in a table, CSPro maps this variable to coordinates 0 through 34, such that age 15 would be row (or column or layer) coordinate 0, age 16 would be coordinate 1, and so on. In post processing the table coordinates, not the values of the variables, are used as the indexes into the table. The table below shows the following two statement as a coordinate:

1. Never married women (layer), age 23 (row), living in urban areas (column) would be:

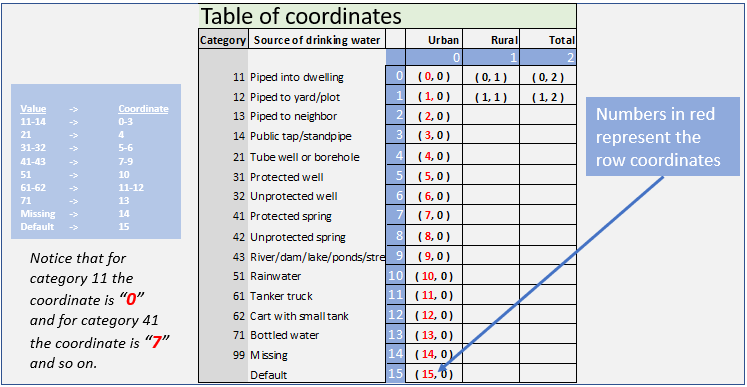
“ x = t1 [ 8, 0, 1 ] ”;

1. Ever married women, age 19 and living on urban areas would be: “x = t1[ 4, 0, 0 ]”;



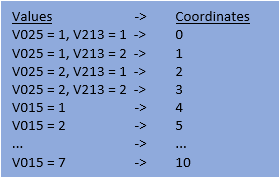
A common error in CSPro is for users to assume that the values of a variable, whose codes start from the value 1, can be used directly as coordinates in table post processing. For example, the variable V015 has the values 1 through 7. However, the coordinates for these values are 0 through 6, as CSPro always numbers the coordinates of a table from 0.

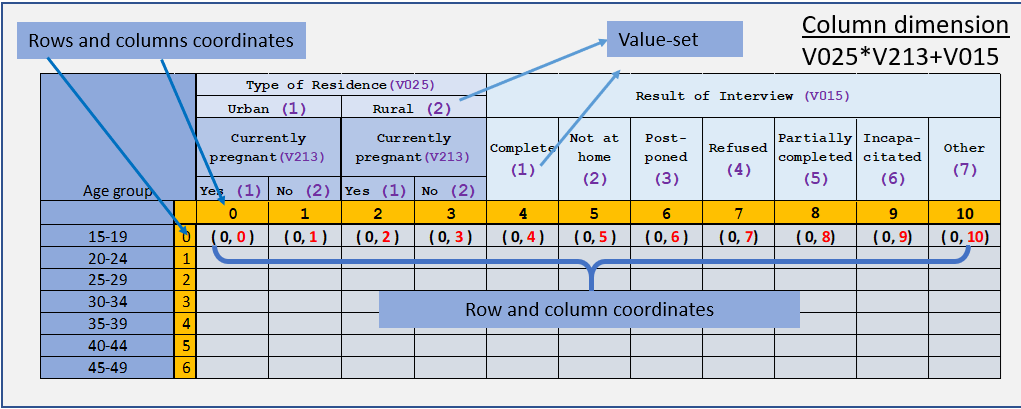
As a further example, a variable like HV201 “Source of drinking water” with valueset ranges defined as 11-14,31,32,41-43,51,61,62,71 and missing, would be translated into table coordinates as follows:



*The missing and default coordinates are included in tables by default, but may be excluded explicitly using EXCLUDE (MISSING, DEFAULT) or EXCLUDE(SPECVAL).*

When the row, column or layer variable lists contain more than one variable, the coordinates of each variable are numbered consecutively following the coordinates of the previous variable. Using an earlier example, the column specification **V025\*V213+V015**, and assuming that all special values are excluded from the table, the coordinate values would be as follows:





If special values are included in the example above, calculating the coordinates to use in the post processing becomes much more complicated. To simplify the work of calculating indexes for tables in postprocessing, CSPro provides several functions that allow the programmer to get the correct indexes into tables by passing the variable name and value as parameters. These functions are described in the following section 7: Post Processing Functions.

# 7. Post Processing Functions

There are several functions that are designed specifically for the postprocessing of a table. These functions can be used only in the dictionary postproc. The postprocessing functions include functions to return table co-ordinates based on the values of variables in rows, columns, and layers, and to facilitate the calculation of percentages, means, medians, and other indicators.

7.1 Functions TBLROW, TBLCOL, and TBLLAY

These functions are used to obtain row, column, and layer indexes by passing table variables, and, optionally, values of those variables, as parameters. They are used to avoid the necessity to use hard coded table co-ordinates.

The syntax of each of the three functions is similar. The main difference between the functions is the dimension of the table that they related to, which is indicated by the last three letters of the function name.

TBLROW will return an integer value of the specific row of a table, TBLCOL will point to a specific column, and TBLLAY will point to a specific layer.

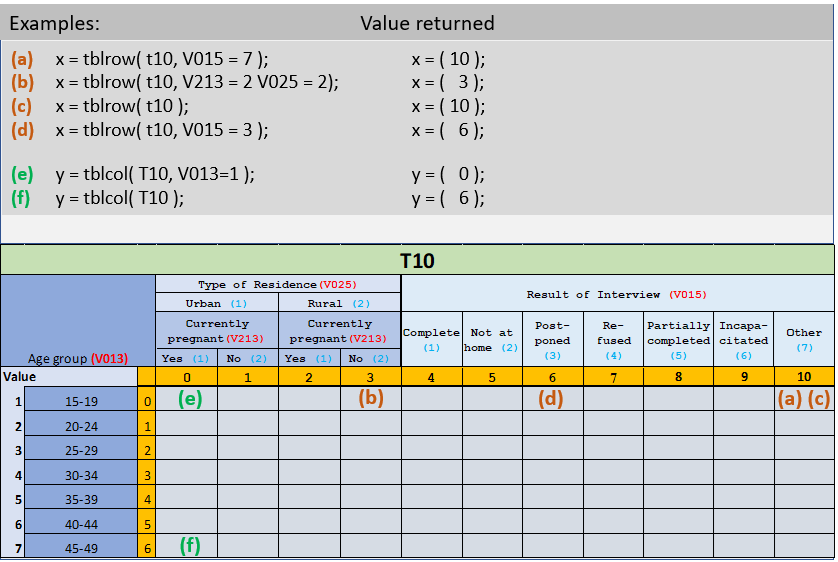
|  |  |
| --- | --- |
|  |  |
|  | *TBLROW / TBLCOL / TBLLAY* | |
|  |  | |
| *Type* | *Function* | |
|  |  | |
| *Return* | *Numeric value* | |
|  |  | |
| *Context* | *Batch (tabulation) applications; dictionary level – postproc* | |
|  |  | |
| *Syntax* | *x =* ***TBLROW(*** *TableName* ***[****VarName=valueset VarName=valueset****]******)****;*  *x =* ***TBLCOL (*** *TableName* ***[****VarName=valueset VarName=valueset****]******)****;*  *x =* ***TBLLAY (*** *TableName* ***[****VarName=valueset VarName=valueset****]******)****;* | |
|  |  | |
| *Description* | *Returns the row, column, or layer coordinate in TableName, where VarName=valueset. If the row, column, or variable list contains nested variables, the order is the number of the sub-table expression, where the plus sign ‘+’ separates sub-tables. Additionally, second VarName=valueset expression may be used to specify the desired coordinate. If VarName is not specified, by default, the order is the last variable or sub-table. Similarly, for any variable, the default coordinate is for the last value of that variable.* | |
|  |  | |

The table below shows some examples of the functions and the value that each example returns. Notice that example (a) and (c) are pointing to the same coordinate.

An important and useful feature of the TBLROW/COL/LAY functions is that the function will always default to returning the highest integer co-ordinate possible, if some parameters are omitted. This is shown in the example table t10 below, where the last column of the table could be returned by each of the following calls to TBLCOL:

|  |
| --- |
| Jtot = tblcol( t10, v015 = 10); { fully specifying the variable name and the value of that variable}  Jtot = tblcol( t10, v015); { omitting the value of the variable returns its highest value}  Jtot = tblcol( t10 ); { just the table name returns the highest value of the last variable in the dimension} |

The DHS standard tables applications make frequent use of this kind of shorthand, so it is important to become familiar with it.

****

7.2 Computing Percentages Using Functions

Continuing with exercise#4, table “Example1”, the cells with the counts of cases are to be replaced by percent distributions (area highlighted in green). The formula for percentage is ***(Numerator / Denominator) \* 100.*** To do this, the rows for each category (numerator) must be divided by the total in the last row (denominator) and multiplied by 100.



**Step 1**: Identify the required row coordinates for the total row. To refer to the row in table Example1 where TOTAL2 = 1, the following TBLROW function can be used:

|  |
| --- |
|  |
| **PROC GLOBAL**  **Numeric itot; { temporary working variable }**  **PROC ZMIR71\_FF**  **Postproc**  **itot = TBLROW( Example1, TOTAL2 = 1); { Last row }** |

The variable **ITOT** will now refer to the row in Example1 where TOTAL2 = 1. The row coordinates where V015 = 1 and V015 = 7 are also needed. These row numbers can be returned with the following statements:

|  |
| --- |
|  |
| **PROC GLOBAL**  **Numeric itot; { temporary working variable }**  **PROC ZMIR71\_FF**  **Postproc**  **Row0 = TBLROW( Example1, V015 = 1);**  **Row7 = TBLROW( Example1, V015 = 7);** |

Now it is possible to write the statements to calculate the percentages. The most direct way to compute these values would be one cell at a time for each column. The wild card (\*) will be used to specify that each line of calculation should be repeated for all cells in a row (all columns):

|  |
| --- |
|  |
| **PROC GLOBAL**  **Numeric itot; { working variables }**  **PROC ZMIR71\_FF**  **Postproc**  **itot = TBLROW( Example1, TOTAL2 = 1);**  **Row0 = TBLROW( Example1, V015 = 1);**  **Row7 = TBLROW( Example1, V015 = 7);**  **{ percentage calculation one cell at the time }**  **Example1 [row0, \* ] = Example1[Row0, \*] / Example1[ itot, \*] \* 100; {All cells of row 0 }**  **Example1 [row0+1, \* ] = Example1[Row0+1, \*] / Example1[ itot, \*] \* 100; {All cells of row 1 }**  **Example1 [row0+2, \* ] = Example1[Row0+2, \*] / Example1[ itot, \*] \* 100; {All cells of row 2 }**  **Example1 [row0+3, \* ] = Example1[Row0+3, \*] / Example1[ itot, \*] \* 100; {All cells of row 3 }**  **Example1 [row0+4, \* ] = Example1[Row0+4, \*] / Example1[ itot, \*] \* 100; {All cells of row 4 }**  **Example1 [row0+5, \* ] = Example1[Row0+5, \*] / Example1[ itot, \*] \* 100; {All cells of row 5 }**  **Example1 [row0+6, \* ] = Example1[Row0+6, \*] / Example1[ itot, \*] \* 100; {All cells of row 6 }** |

Writing the statements in the example above is syntactically correct but is not the most efficient means of doing so. For this particular example where there is only one row-variable with seven categories (V015), the statement could be repeated seven times. However, this approach will not scale well for bigger tables or applications where there are many tables.

Another CSPro command that can accomplish the same result with one set of statements is the **DO-WHILE** loop. The do-while statement executes the statement repeatedly, in a loop, until the logical condition is true. Only two working variables are needed: *itot* and *i*. ***itot*** will mark the last row of the table and “i” will move thru each row starting with the first one (row 0 coordinate).

|  |
| --- |
|  |
| **PROC GLOBAL**  **Numeric i, itot; { temporary working variables }**  **PROC ZMIR71\_FF**  **Postproc**  **itot = TBLROW( Example1, TOTAL2 = 1);**  **{ percentage calculation using loops }**  **do i = 0 while i < itot**  **Example1[i,\*] = Example1[i,\*] \* 100 / Example1[itot,\*];**  **enddo;**  ***{ note: Using the wild code (“\*”), the computation of percentages can be simplified even more }*** |

The variable “i” will be initialized with the value zero **“do i = 0”**, and for each iteration “i” will increment by one until the value of “i” is less than the value of variable “itot” **while i < itot**. Essentially, it will start with the table row where V015 is equal to 1 and calculate the percentage in each column using the table cell where TOTAL2 is equal to 1 as the denominator. It will then increment the value of i and do the same thing for each subsequent row, until it reaches the table row where V015 is equal to 7.

|  |
| --- |
|  |
| **{ percentage calculation using loops }**  **do i = 0 while i < itot**  **When “i = 0” {iteration 1}**  **Example1[0,\*] = Example1[0,\*] \* 100 / Example1[itot,\*];**  **When “i = 1” {iteration 2}**  **Example1[1,\*] = Example1[1,\*] \* 100 / Example1[itot,\*];**  **When “i = 2” {iteration 3}**  **Example1[2,\*] = Example1[2,\*] \* 100 / Example1[itot,\*];**  **...**  **...**  **When “i = 7” {iteration 8}**  **The loop will stop since condition “i < itot” is no longer true**  **enddo;** |

|  |
| --- |
| **Exercise #5a – Post Processing Functions Test** |
| * *The table printed below shows all the information needed for this test: table name, variable names, value labels and value set, CSPro coordinates. The main objective is to create the logic that will populate the row highlighted in light gray color.*   ***Note: temporary working variables’ names are as fallow****:* For rows => i1524, i1519, i2024  *Using the table* **T3021** *below, write the following statements:*   1. *Using Function TBLROW, write the logic that return the coordinate for age group 15-19:*   *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*   1. *Using Function TBLROW, write the logic that return the coordinate for age group 20-24*   *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*   1. *Using Function TBLROW, write the logic that return the coordinate for age group 15-24*   *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*   1. *Using the coordinates defined in point 1), 2) and 3) write the statement that summarized the row “Age 15-19” for all cells, plus the row “Age 20-24” for all cells into the row “Age 15-24” for all cells.*   *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |

|  |
| --- |
| **Exercise #5b** |
| * *Using the application “MyBatchTables.bch”* * *Add the logic to calculate the percent distribution of table “example 1”* * *Use the two approaches: a) use separate statements to calculate cell by cell b) use a do-while loop to achieve the same result* * *Run the application* * *Use tables-editor to export the table to RTF* * *Verified that the output is correct* * *Table should look like the table below*  |  |  |  |  | | --- | --- | --- | --- | | Example 1.Results of individual interviews | | | | | Percent distribution of results of individual interviews by urban/rural residence Zambia DHS 2018-19 | | | | |  | Type of place of residence | |  | |  | Urban | Rural | Total | | **Result of individual interview** |  |  |  | | Completed | 95.6 | 97.0 | 96.4 | | Not at home | 2.9 | 2.0 | 2.4 | | Postponed | 0.0 | 0.0 | 0.0 | | Refused | 0.9 | 0.4 | 0.6 | | Partially completed | 0.0 | 0.0 | 0.0 | | Respondent incapacitated | 0.3 | 0.4 | 0.4 | | Other | 0.3 | 0.1 | 0.1 | |  |  |  |  | | Number | 5,766 | 8,423 | 14,189 | |

7.3 TBLSUM Function

This function returns the sum of a range of rows, columns, or layers of a table, called TBLSUM. The syntax of TBLSUM is as follows:

|  |  |
| --- | --- |
|  |  |
|  | *TBLSUM function* |
|  |  |
| *Type* | *Function* |
|  |  |
| *Return* | *Numeric value* |
|  |  |
| *Context* | *Batch (tabulation) applications; dictionary level – postproc* |
|  |  |
| *Syntax* | *x =* ***TBLSUM(*** *domain, TableName****[ row:row, column:column, layer:layer ] )****;* |
|  |  |
| *Description* | Calculates the sum of a range of cells in the rows, columns, or layers of a table. There are three domains: ROW, COLUMN, or LAYER. |
|  |  |
| *Example* | x = TBLSUM( ROW t10[ 0:6, \* ]);  This example will return the sum of values in (the domain) row 0 to 6 of table t10. The asterisk represents “all” columns. There are not layers specified since t10 is a two dimensions table. |
|  |  |
|  | *Note: This function is extensively used in the production of DHS final report tables. Specifically, on percent distribution tables where all the parts should add up to a 100 percent. The most efficient way to use this function is in combination with the row, column, and layers coordinates functions described above.* |

Below is an example of the use of the TBLSUM function. The sub-table highlighted in yellow will receive the sum of the percent distributions in ” (column 1 + column 2).

*The examples below show two approaches on how to point to the column in yellow: a) by writing the exact column coordinate number, b) by using the function TBLCOL.*

|  |
| --- |
|  |
| **A) PROC DICTIONARY\_FF**  **Postproc**  **Table1[ \*,2 ] = TBLSUM*( domain-? TableName[ row-?, column-? ] );*** |
|  |
| **B) PROC GLOBAL**  **Numeric x; { *temporary working variable “x” should be declared as numeric* }**  **PROC DICTIONARY\_FF**  **Postproc**  **x = TBLCOL( Table1, Var2 = 0 ); {*“0” is the valueset for TOTAL}***  **Table1[ \*, x ] = TBLSUM*( domain-? TableName[ row-?, column-? ] );***  ***{ NOTE: This approach is more precise and efficient and prompt for less error.}*** |

**Step 2**: What is the domain? The domain is the row, column or layers that need to be summarized. Since columns will be added, the domain for table1 is COLUMN.

|  |
| --- |
|  |
| **x = TBLCOL( Table1, Var2 = 0); {*“0” is the valueset for TOTAL}***  **Table1[ \*, x ] = TBLSUM*( COLUMN TableName[ row-?, column-? ] );***  ***{ NOTE: This approach is more precise and efficient and prompt for less error.}*** |

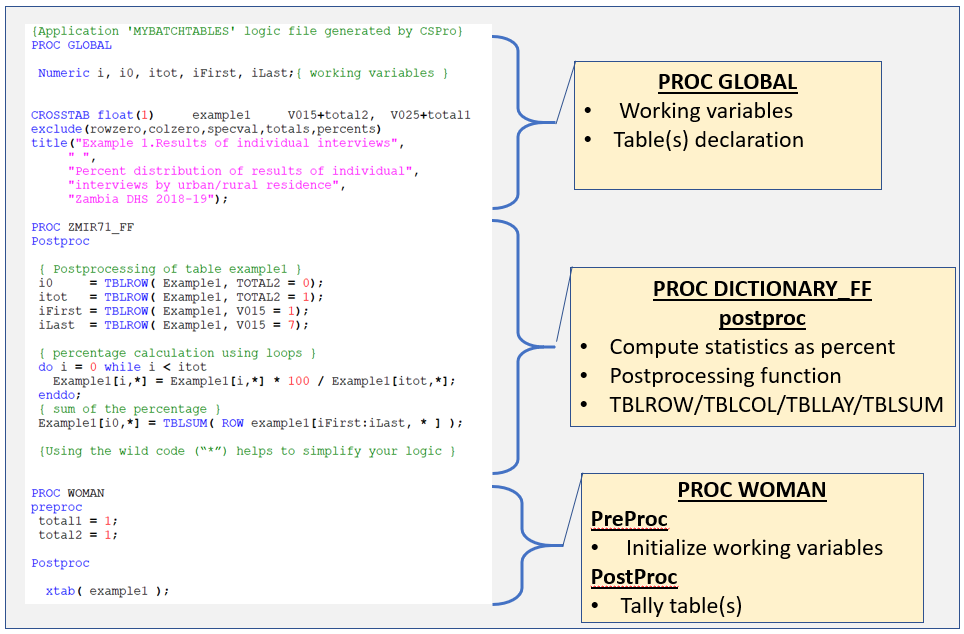
**Step 3:** Which are the cells to be summed? Table1 shows that there are 2 columns that need to be added: var1-category 1 and var1-category 2. Since the two columns for **“all”** the rows (residence and wealth quintile) will be added; then the wildcard character “\*” should be used to specify the row coordinates. There is different approach that can be used to specify the columns to be added: a) by using coordinates or b) by using the exact range of columns. For this second approach it is highly recommended to check in the dictionary the valueset of all variables which columns will be added to avoid missing any column.

|  |
| --- |
|  |
| **A) x = TBLCOL( Table1, Var2 = 0); {*“0” is the valueset for TOTAL}***  **Table1[ \*, x ] = TBLSUM*( COLUMN Table1[ \*, 0:1 ] );***  ***{ NOTE: the range 0:1 is referencing to the CSPro coordinates, not to the value set (1:2) }*** | |
|  |
| **B) PROC GLOBAL**  **Numeric x, FirstC, LastC;{ working variables should be declared as numeric }**  **PROC DICTIONARY\_FF**  **Postproc**  **x = TBLCOL( Table1, Var2 = 0); {*“0” is the valueset for TOTAL}***  **FirstC = TBLCOL( Table1, Var1 =1); { First column *}***  **LastC = TBLCOL( Table1, Var1 =2); {Last column *}***  **x = TBLSUM*( COLUMN, Table1[ \*, FirstC : LastC ] );*** | |
|  |
|  | |

|  |
| --- |
| **Exercise #6a – Post Processing Functions Test** |
| * *The table printed below shows all the information needed for this test: table name, variable names, value labels and value set, CSPro coordinates. The main objective is to create the logic that will populate the column highlighted in dark gray color.*   ***Note: temporary working variables’ names are as fallow****:* For column => j0, j5, j6  *Using the table* **T3021** *below, write the following statements:*   1. *Using function TBLCOL write the logic that return the coordinates for: a) “No education”, b) higher education, and c) Total.*   *a)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *b)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *c)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*   1. *Using the working variables defined in point 5) and the Function TBLSUM, write the statement that will add up the values in columns 0 to 5.*   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

|  |
| --- |
| **Exercise #6b** |
| *There is one final step to finalize the table “example 1”. Since this table is a percent distribution, all rows need to be added for each column using the function TBLSUM. The sum should be equal to 100. The table below shows the cells in yellow color that need to be populated with the sum of the percentages.*    *Instructions:*   * *Because variable TOTAL2 has only one category “1= Number”, go to the working dictionary and add a new category to this variable “0 = Total percent”. This category will be used as the coordinate where the summarization of the percentages will be displayed.*   *TOTAL2 will now have two valueset:*   * + - *0 = Total percent*     - *1 = Number* * *Go to the logic to “PROC ZMIR71\_FF”, postproc to write the statements*    + *Define the coordinate that point to the row receiving the sum of the rows*     - ***i0*** *= TBLROW( example1, TOTAL2=0);*   + *Define the coordinate that return the first row to be added (V015 = 1)*     - ***iFirst*** *= TBLROW( example1, V015 = 1);*   + *Define the coordinate that return the last coordinate to be added (V015=7)*     - ***iLast*** *= TBLROW( example1, V015 = 7);* * *Using the coordinates i0, iFirst, and iLast, define the statement that will add the range of rows*   + *Example1[****i0,****\*] = TBLSUM( ROW example1[****iFirst:iLast****, \* ] );*   *{This statement will set the “Total Percent” row equal to the sum of the rows from row “iFirst" (V015 = 1) to row “iLast” (V015 = 7)}*   * *Run your application and verify that the table “example1” displays the sum of the percentage as “100.0”* |

The final code for table “example1” is therefore as follows:

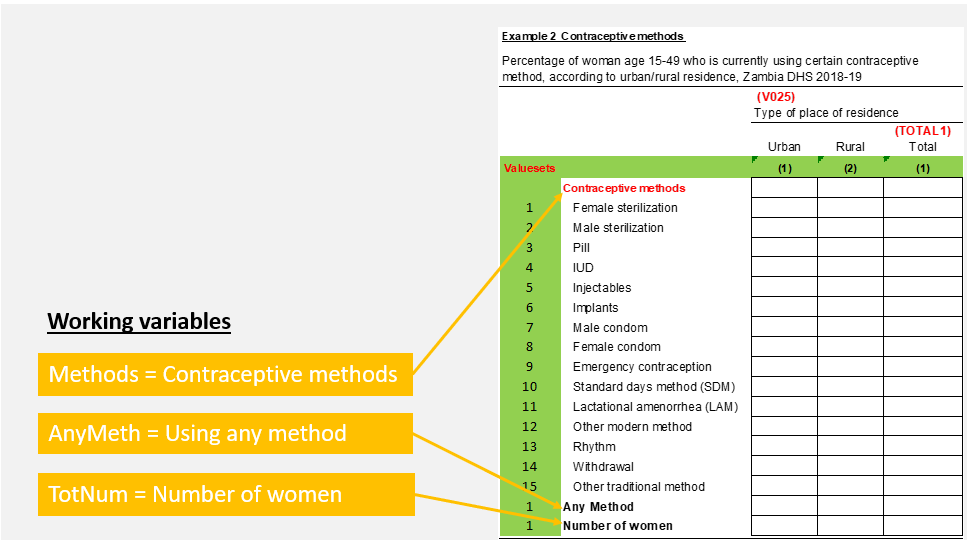


# 8. Tabbing Multiple Times

For some types of tables, a useful technique is to call the XTAB function multiple times for the same Table. Most often, this technique works well for tables that are not simple percent distributions.

To illustrate, consider the following example.

A researcher is interested in the percentage of women who is currently using certain contraceptive methods (based on Section 31 (REC31) of the recode file) broken down by urban/rural residence. This table will not be a percent distribution, since some women may be currently using more than one method combined. The table that the researcher wants to construct is structured as follows:



V307 (the variable for “Type of method currently using”) cannot be used directly because CSPro does not allow the use of Multiple Variables or Variables in Multiple Sections in tables. Therefore, the programmer will create three working storage variables for the row dimension of the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Variable Label** | **Length** | **Valueset** |
| **METHODS** | Contraceptive Method | 2 | 1-15 as table above |
| **ANYMETH** | (“.”) | 1 | 1= Any Method |
| **TOTNUM** | (“.”) | 1 | 1 =Number of women |

For the column dimension there are two variables: **V025** and **TOTAL1**. Both variables are already defined; V025 is part of the data dictionary and **TOTAL1** was added to the working storage for previous table “example1”. It can be reused for our current table.

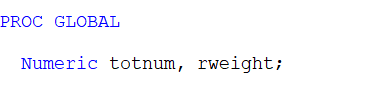
The commands to create and fill the desired table can be placed in the same application “MyBatchTables”, alongside the commands which created the previous table “example1”. The following Table crosstab command defines the structure of the table, which is named “example2”.

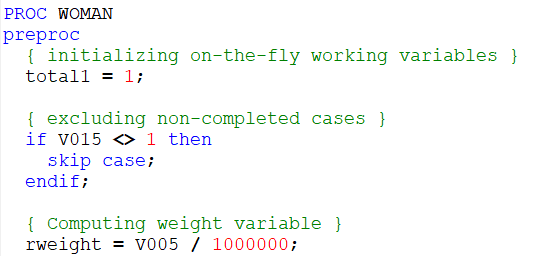
|  |
| --- |
|  |
| **Crosstab float(1) example2 Methods+AnyMeth+TotNum V025+TOTAL1**  **exclude(rowzero,colzero,specval,totals,percents)**  **title("Example 2. Contraceptive methods”,"",**  **"Percent of women who is currently using specific methods of”,**  **"contraception, or any method of contraception, by”,**  **"urban/rural residence, Zambia DHS 2018-19" );** |

There are 2 new steps that need to be implemented to compute this table: 1) non-complete questionnaires should be excluded, and 2) the weight factor should be applied (see section 10 of the tables in batch manual).

* + The SKIP-CASE command will be used to exclude non-complete questionnaires (based on V015). This command should be placed on the PreProc of the “PROC WOMAN” level. This command can only be used in the PreProc of a Level Procedure
  + The weight variable (V005) has 6 implicit decimals, meaning that there is no decimal point (“.”) in the dataset. In order to add the 6 decimals, the variable should be divided by 1 million. It is necessary to create a temporary “on-the-fly working variable named “rweight” to compute the weight. This variable will be assigned on the PreProc of the “PROC WOMAN” level.
  + The variable **TOTAL1** was initialized for the previous table so it is not necessary to do it again.

All of these tasks are accomplished in the following syntax:



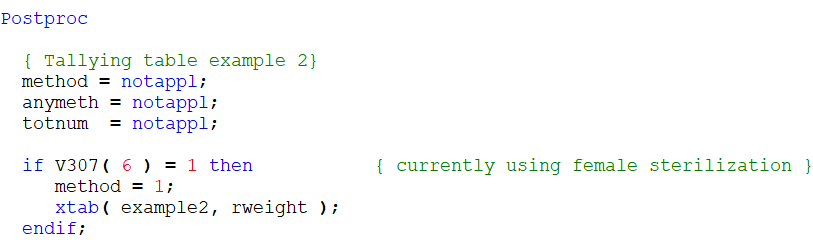


All of the **xtab** functions will be in PROC WOMAN PostProc because the appropriate cells of the table should be tallied once for each respondent. More precisely, the appropriate cells should be tallied once for each method a woman is currently using, based on the values for each occurrence of V307( ).

When writing the code for tables where there are multiple tallies to the same table for the same case, the programmer needs to carefully manage assignment of values to variables in the table to avoid unwanted tallies to cells in that table. Usually, in a table constructed with multiple tallies, there will be one dimension which remains “fixed”, that is, is always tallied, and another dimension where the values of variables are changed for each multiple tally. In our example, the “fixed” dimension is the column dimension containing the urban-rural (V025) and the total (total1) variables. For the row dimension, we have three variables METHODS, ANYMETH, and TOTNUM, which will be tallied separately using multiple calls to the XTAB function. The values of all three variables must be carefully managed, as described below, to ensure that only one variable is tallied at a time.

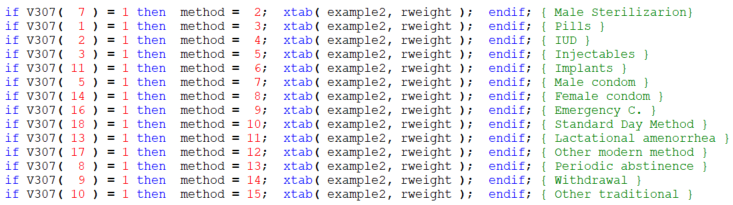
Before tallying, the working variables METHODS, ANYMETH, and TOTNUM must be initialized to NOTAPPL. This has to be done so that the initial values are reset for each respondent[[1]](#footnote-2). If initial values are not reset for each woman, the values will be retained from the previous woman. Also, since we will begin by multiple tallying to the METHODS variable, we need to prevent any unwanted tallies to ANYMETH and TOTNUM.

For the first method “Female Sterilization” in table *example2*, occurrence V307(6), the statements could be as follows.



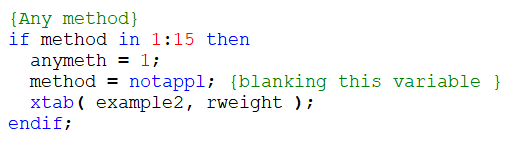
These statements will increase the value of the cell in the table where Method = 1 and V025 is the value from the data file (urban or rural) by an amount equal to RWEIGHT for the current woman. Notices that METHOD is the only variable that is tallied. ANYMETH and TOTNUM are not tallied because their values were set to NotAppl.

Similar statements could then be written for the second and so on for all fifteen methods:

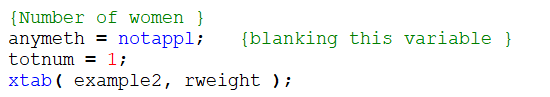


This will result in 15 IF..THEN..ENDIF commands, each with its own Xtab function. It is important to highlight that the order of contraceptive methods in the table is different from the occurrence on V307() so in order of tallying the information in the proper contraceptive label, an extra step of verifying the right occurrence for each method on V307() should be done.

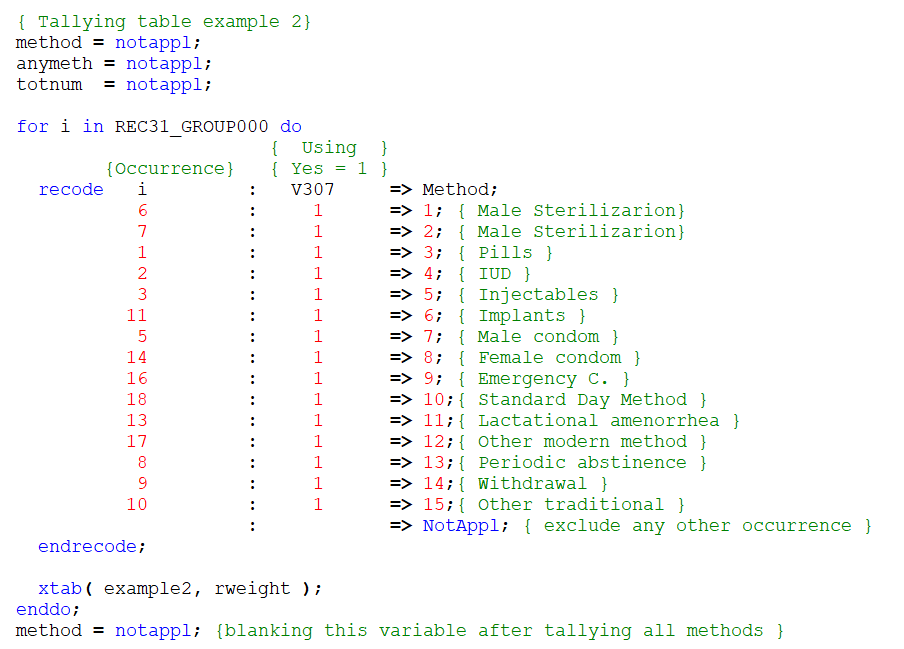
After all contraceptive have been tallied, the next is to tally the row “Any method”. The value assigned to variable METHOD can be used to define if the respondent has used any method. After the 15 statements, if METHOD has a valid value, means that the respondent has used at least one. Notice that before using the Xtab function to tally “ANYMETH”, the variable METHOD should be set back to NotAppl to prevent double counting the last method used. See the logic below.



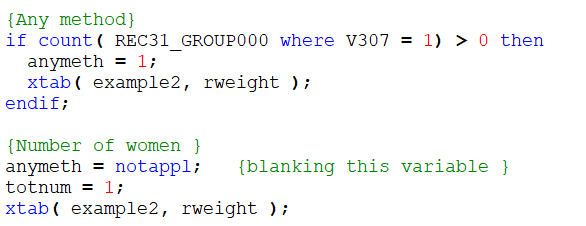
The final step is to increment the Row for the total number of women for every woman (TOTNUM). To prevent duplicating the number of women using any method, the same principle of setting ANYMETH to NotAppl should be followed. See logic below.



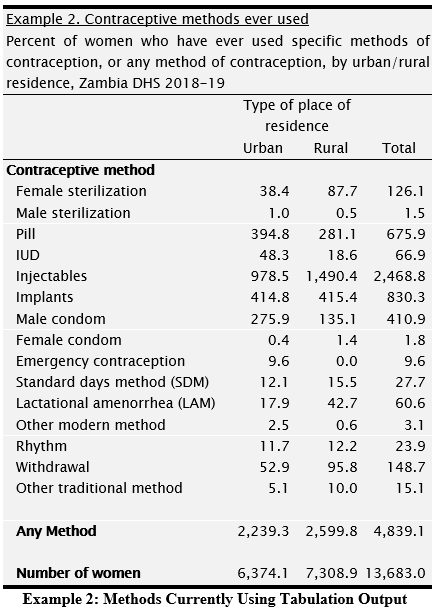
Even though these statements produce the expected result, this can be accomplished more quickly with the following FOR loop command. The FOR command will execute the statement(s) for each occurrence of contraceptive method ( multiple record or group). RECODE-ENDRECODE command is used to reclassify the valueset of each contraceptive method in V307 into the working variable METHOD.



There will be some variation to tally ANYMETH. In the data dictionary ZMIR71.DCF, the multiple group REC31\_GROUP000 has 20 occurences but only 15 of them where asked. The last line in the RECODE statement set to NotAppl any occurrence not asked. This will prevent to increment the last method used for each case. The **COUNT** function is a simple command that will be used to set ANYMETH. The format of this command is *“i =* ***count****(multiple-item [****where*** *condition]);”* In batch application, this function will always return the total number of occurrences of a multiple record or group. If the “*[****where*** *condition]”* is included , the function will return the number of occurrences for which the condition is true.Notice that the statement below includes an extra condition **“> 0”** to convert the final output in two possible values “1=true/0=false” (Boolean); saying that, if the count of “yes” is greater than zero, the condition is true and the return will be “1” and if it is equal to zero, the condition is false and the return will be“0”. The IF statement below will only consider when the return is true.



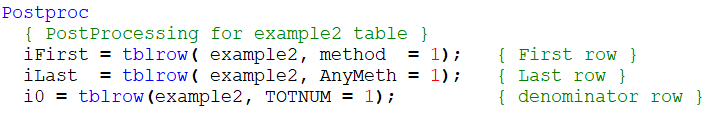
If the Application is written and executed as described so far, it will produce the Table shown below:



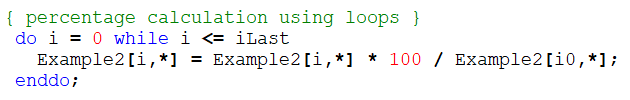
8.1 Post Processing

There are two tasks remaining. First, all of the cells in the example2 table (except on the Row for Number of Women) must be calculated as percents of the numbers of women in the respective Columns. Second, the Number of Women should be rounded off to zero decimal places. The first of these tasks must be done in the PROC DICTIONARY\_FF **Postproc** because they are to be performed after all data are processed. The rounding can be done with TablesEditor.

To calculate the percentages, each of the Rows for the Variable METHOD should be divided by the “Number of Women” Row (where TOTNUM = 1). The first step is to locate the Rows with the minimum and maximum values for METHOD and TOTNUM:

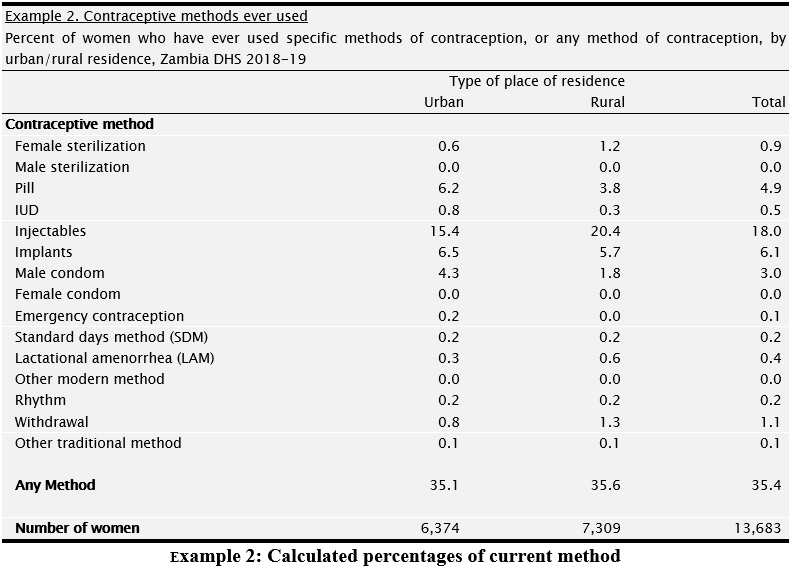


Percents can then be calculated by dividing each METHOD row by the “Number of Women” Row:



To round the numbers of women in the “Number of Women” Row. Open the “TablesEditor” utility and set to “no decimals” for the last row. (See tables editor section).

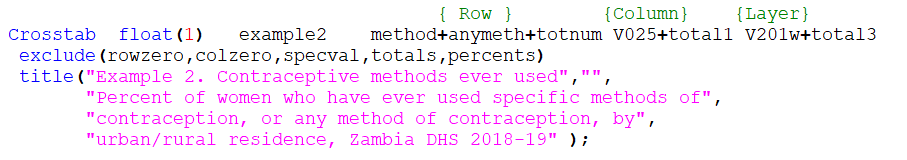
This will produce the following Table:



8.2 Creating Tables with Layers

Now suppose that the information in the EXAMPLE2 table should be given separately for women who have had children and those who have not (based on the response to V201). This can be accomplished by using Layers[[2]](#footnote-3). Two working variables will be added as a third dimension (layers) to the table declaration: V201w+TOTAL3.

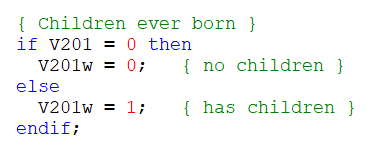
To use Layers, add a list of Layer Variables to the Crosstab Command. The Layer Variables should be placed after the list of Column Variables. The following table definition (a modification of the existing table definition) will create the table structure described above:



The TOTAL3 and V201w Variables will have to be defined in the working storage. TOTAL3 will provide a “total” Layer which will include all women. Define the variable label, value label and length as follow:

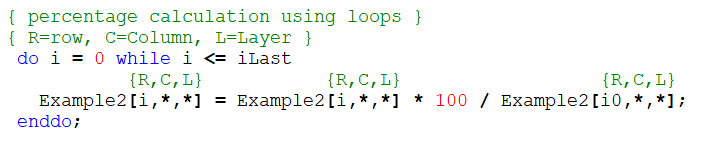
|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Variable Label** | **Length** | **Valueset** |
| **V201w** | (“.”) | 1 | 0 “Has Had children  1 “Has Never Had Children |
| **TOTAL3** | (“.”) | 1 | 1= All Women |
|  |  |  |  |

Since TOTAL3 will accumulate the number of women, it should be initialized in the PreProc of the dictionary level. V201w will also have to be assigned in PROC WOMAN, e.g.:



The PROC DICTIONARY\_FF PostProc, however, will require some minor changes due to the addition of another dimension (Layer). The number of dimensions used in the PROC DICTIONARY\_FF PostProc must be consistent with the number of dimensions defined in the crosstab Command[[3]](#footnote-4). A Table that uses only Rows and Columns has 2 dimensions. A table that uses Rows, Columns, and Layers has three dimensions.

The first step is to change the calculation of percents to consider Layers. These calculations can be modified by adding “\*” wild code in the third dimension of table references.



*( In this context, the “\*” wild code will refer to all Layers. The result will be that percentages will be calculated in the same way for all Layers)*

The rounding of the numbers of women need to be done using the Tables Editor as explained in 1.12 and can thus be ignored in the tabulation application.

This updated syntax will produce three sub-tables, one for each Layer. The table is displayed below.

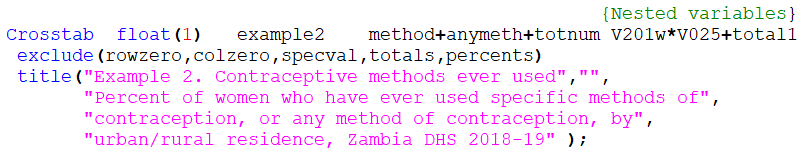
|  |  |  |  |
| --- | --- | --- | --- |
| Example 2. Contraceptive methods ever used | | | |
| Percent of women who have ever used specific methods of contraception, or any method of contraception, by urban/rural residence, Zambia DHS 2018-19 | | | |
|  | Type of place of residence | |  |
|  | Urban | Rural | Total |
| HAS HAD CHILDREN | | | |
| **Contraceptive method** |  |  |  |
| Female sterilization | 0.0 | 0.0 | 0.0 |
| Male sterilization | 0.0 | 0.0 | 0.0 |
| Pill | 0.6 | 0.3 | 0.5 |
| IUD | 0.0 | 0.0 | 0.0 |
| Injectables | 0.6 | 1.0 | 0.8 |
| Implants | 0.3 | 0.4 | 0.3 |
| Male condom | 4.0 | 1.6 | 3.0 |
| Female condom | 0.0 | 0.0 | 0.0 |
| Emergency contraception | 0.3 | 0.0 | 0.2 |
| Standard days method (SDM) | 0.0 | 0.0 | 0.0 |
| Lactational amenorrhea (LAM) | 0.0 | 0.0 | 0.0 |
| Other modern method | 0.0 | 0.0 | 0.0 |
| Rhythm | 0.0 | 0.1 | 0.0 |
| Withdrawal | 0.2 | 0.0 | 0.1 |
| Other traditional method | 0.0 | 0.0 | 0.0 |
|  |  |  |  |
| Any Method | 6.0 | 3.5 | 5.0 |
|  |  |  |  |
| Number of women | 1,914 | 1,450 | 3,365 |
| HAS NEVER HAD CHILDREN | | | |
| **Contraceptive method** |  |  |  |
| Female sterilization | 0.9 | 1.5 | 1.2 |
| Male sterilization | 0.0 | 0.0 | 0.0 |
| Pill | 8.6 | 4.7 | 6.4 |
| IUD | 1.1 | 0.3 | 0.6 |
| Injectables | 21.7 | 25.2 | 23.7 |
| Implants | 9.2 | 7.0 | 7.9 |
| Male condom | 4.5 | 1.9 | 3.0 |
| Female condom | 0.0 | 0.0 | 0.0 |
| Emergency contraception | 0.1 | 0.0 | 0.0 |
| Standard days method (SDM) | 0.3 | 0.3 | 0.3 |
| Lactational amenorrhea (LAM) | 0.4 | 0.7 | 0.6 |
| Other modern method | 0.1 | 0.0 | 0.0 |
| Rhythm | 0.3 | 0.2 | 0.2 |
| Withdrawal | 1.1 | 1.6 | 1.4 |
| Other traditional method | 0.1 | 0.2 | 0.1 |
|  |  |  |  |
| Any Method | 47.6 | 43.5 | 45.3 |
|  |  |  |  |
| Number of women | 4,460 | 5,859 | 10,318 |
| ALL WOMEN | | | |
| **Contraceptive method** |  |  |  |
| Female sterilization | 0.6 | 1.2 | 0.9 |
| Male sterilization | 0.0 | 0.0 | 0.0 |
| Pill | 6.2 | 3.8 | 4.9 |
| IUD | 0.8 | 0.3 | 0.5 |
| Injectables | 15.4 | 20.4 | 18.0 |
| Implants | 6.5 | 5.7 | 6.1 |
| Male condom | 4.3 | 1.8 | 3.0 |
| Female condom | 0.0 | 0.0 | 0.0 |
| Emergency contraception | 0.2 | 0.0 | 0.1 |
| Standard days method (SDM) | 0.2 | 0.2 | 0.2 |
| Lactational amenorrhea (LAM) | 0.3 | 0.6 | 0.4 |
| Other modern method | 0.0 | 0.0 | 0.0 |
| Rhythm | 0.2 | 0.2 | 0.2 |
| Withdrawal | 0.8 | 1.3 | 1.1 |
| Other traditional method | 0.1 | 0.1 | 0.1 |
|  |  |  |  |
| Any Method | 35.1 | 35.6 | 35.4 |
|  |  |  |  |
| Number of women | 6,374 | 7,309 | 13,683 |

**Example 2: Current method with layers**

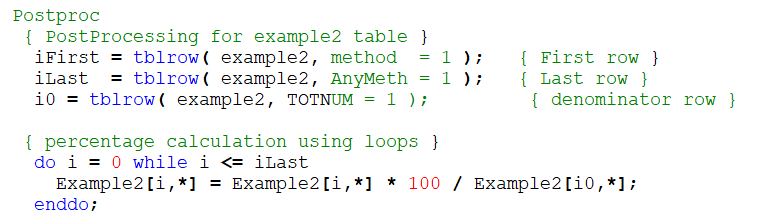
# 

# 9. Creating Tables with Nested Variables

Nesting can create the table above without using Layer Variables. This is done by expanding the columns using the “\*” convention. If Variable1 in a Table’s Row or Column[[4]](#footnote-5) is separated from Variable2 by a “\*” instead of a “+” then Variable2 is said to be “nested” under Variable1. CSPro will construct a table with a set of Columns (or Rows) with all values of Variable2 for each value of Variable1. The following example shows urban/rural residence (V025) nested under ever had children (v201w):

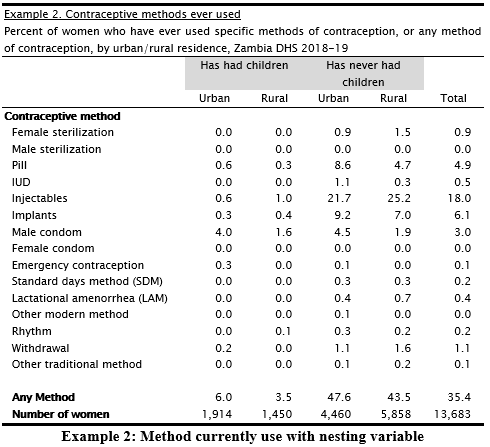


Note that the statement for TBLROW involving nested columns should specify values for both variables unless the default is desired. For this particular table, the nested variable is in the column dimension and since the percent is calculated based on the **“i0”** row, the TBLROW assignment of **i0**, **iFirst** and **iLast** do not need to be modified to add v201w:

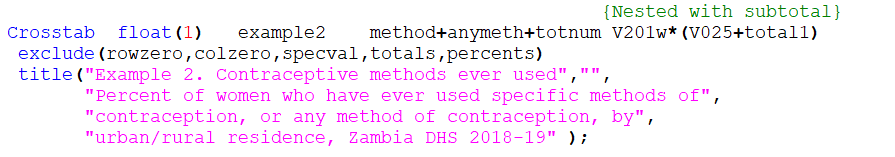


In the above example, CSPro will create a Table with urban and rural Columns for each value of V025.

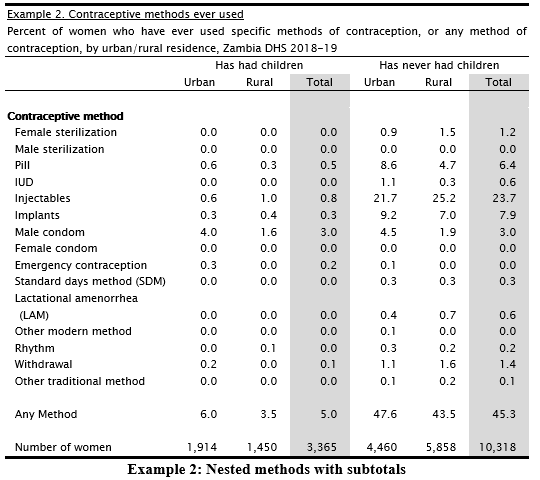
This table would have the following structure:



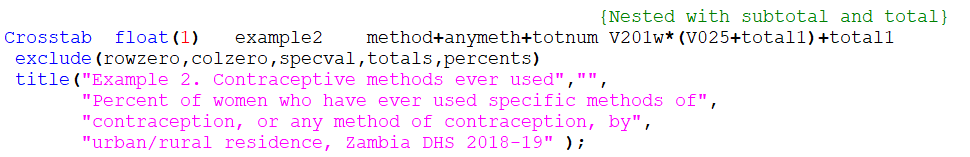
The table definition can be modified to include subtotals for the values of V025 by using parenthesis:

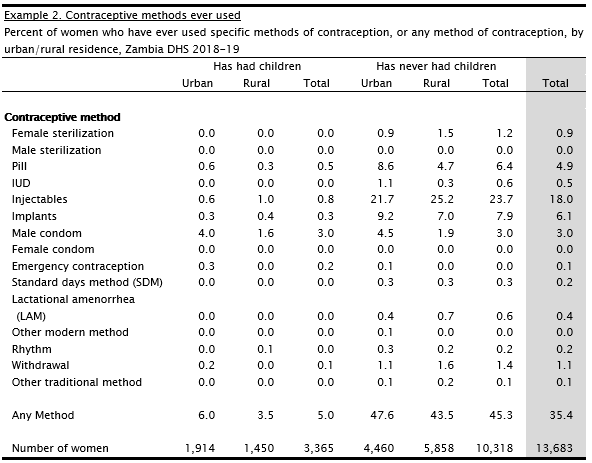


This will produce the following table:



Another addition to the table definition can be to include “total” for the values of V025 by adding the same TOTAL1 variable after the parenthesis with a “+” sign:





In all three of the above nesting examples, there are 2 dimensions in the Table[[5]](#footnote-6), so the PROC DICTIONARY\_FF **PostProc** will contain the statements that were used before Layers were introduced. In this example, nesting has two advantages over layering. First, it produces the desired output in one table, as opposed to three. Second, it does not require modification of the existing post processing logic.

# 10. Weights[[6]](#footnote-7)

All Demographic and Health Surveys are based on a sample that is randomly selected to be representative of the target population. The main target population used by the DHS are country as a total, urban/rural and the first administrative country division i.e. Region (sometimes this can be represented by the second administrative division, i.e. District). The sample is usually drawn from the sample frame of the most recent census. The sample frame is a list of all EAs that covers entirely the target population.

DHS surveys are household-based surveys. Once all the households in the DHS sample are selected and data is collected, the sample weights will be calculated, and all analysis of DHS data should use the sample weights. The weights are calculated for each household and individual interviewed.

Basically, the sample weight inflates or deflates the population selected to extrapolate the sample to the target population. There are three main reasons for which it is necessary to use sampling weights:

1. To make inferences that are statistically valid
2. To correct the non-response or other non-sampling errors
3. To keep the weighted sample distribution closed to the target population distribution.

10.1 Sampling Weights and Design Weights

The design weight of a household or an individual woman or men (the sampling unit) is the inverse of the overall probability of selection of the unit in the sample. The sampling weight of a household or individual is the design weight corrected for non-response or other adjustments.

For all DHS surveys, both, the design weight and the sampling weight are calculated for every household and/ or individual interviewed.

10.2 Normalization of Sampling Weights

The procedure of normalizing the weights is just making the number of unweighted cases to match with the number of weighted cases at the national level for households and individuals. Normalization is not necessary, but it is a DHS practice to avoid having final report tables where the number of weighted cases are very large numbers. Normalizing the standard weight is very simple, just by multiplying the sample weight by a unique constant at the national level. The constant is the number of completed cases divided by the total number of weighted cases.

10.3 Main Sampling Weights in DHS Data

On every DHS survey, there are 2 main sampling weights: Household weights (HV005) and individual weights (V005) for women 15-49. If the survey includes men’s interview there will be two more additional weights: household weight for men’s subsample (HV028) and individual men weight (MV005). Noticed that if men’s interviews will be conducted on all the households selected (no subsample) it is not necessary to calculate HV028, instead the main household weight can be used (HV005).

|  |  |  |
| --- | --- | --- |
| **Sampling Weights** | **Variable Name** | **Description** |
| **Household weight** | HV005 | The household weight is the inverse of its household selection probability multiplied by the inverse of the household response rate in the stratum. |
| **Household weight for male subsample** | HV028 | The household weight for the men’s subsample is the inverse of its household selection probability for the subsample multiplied by the inverse of the household response rate for the subsample in the stratum. |
| **Individual women weight** | V005 | The individual weight for women is the household weight (HV005) multiplied by the inverse of the individual response rate for women in the stratum. |
| **Individual men weight** | MV005 | The individual weight for men is the household weight for the men’s subsample (HV028) multiplied by the inverse of the individual response rate for men in the stratum |

10.4 Domestic Violence (DV) and HIV Sampling Weights

If the survey includes the Domestic Violence (DV) module and/or HIV testing, then additional weights will be calculated for use with the DV data (D005) and/or HIV data (HIV05), due to the differential probability in selecting the subsample for this unit of analysis and the non-response adjustment. In the case of DV, for most of DHS surveys, only one woman per sampled household is selected.

In the case of HIV, the weights are calculated separately for women and men by correcting the household sampling weight for the non-response rates of women and men to the HIV testing. HIV test result data for women and/or men are stored is a single dataset that it is not part of the main individual data. The methodology used to calculate the HIV weights for women and men are the same. The only difference is that the data is normalized for women and men together.

10.5 Application of Sampling Weights in CSPro

A standard practice of the DHS data archive is to represent all numeric data as integer values in distributed datasets. Since the sample weights variables, HV005, HV028, V005, MV005, D005, HIV05, are calculated to six decimals, to obtain the six decimal place it is necessary to divide each of these variables by 1,000,000 before using them. In CSPro, the weight is applied as an optional parameter when calling the XTAB function (see page 15 for more detailed information on the XTAB command syntax):

**Step 1**: Declare a numeric temporary variable in the PROC GLOBAL

|  |
| --- |
| PROC GLOBAL  Numeric HHWeight; { temporary working variable } |

**Step 2**: Assign value to HHWeight in the preproc of the appropriate level

|  |
| --- |
| PROC HOUSEHOLD  Preproc  { Computing weight variable }  HHWeight = HV005 / 1000000; |

**Step 3**: Apply weights to data when tallying

|  |
| --- |
| PROC HOUSEHOLD  Postproc  XTAB( Table\_Name, HHWeight); |

1. Table’s variables that have a valueset of NotAppl will be excluded from the table because special values (SPECVAL) are excluded as part of the CROSSTAB Command – EXCLUDE( ). [↑](#footnote-ref-2)
2. The next section will show how to accomplish the same task using nesting. [↑](#footnote-ref-3)
3. Where Rows, Columns, and Layers are considered dimensions. [↑](#footnote-ref-4)
4. This convention cannot be used for Layers. [↑](#footnote-ref-5)
5. The second parameter of the TBLCOL Function is the number of the Variable in the list of Column Variables. When an asterisk “\*” is used all of the Variables in that group of Variables is counted as one Variable in TBLCOL. [↑](#footnote-ref-6)
6. For more information about DHS sampling weights calculation procedures visit our [Guide to DHS Statistics](https://www.dhsprogram.com/Data/Guide-to-DHS-Statistics/index.cfm) – go to the Section on “Analyzing DHS Data” in Chapter 1 or by searching for “Analyzing DHS Data”. Also check our videos on sampling: [DHS Program YouTube videos](https://www.youtube.com/playlist?list=PLagqLv-gqpTMx2Q10C_prJRnCtM55C0o8), [Introduction to DHS Sampling Procedures](https://www.youtube.com/watch?v=DD5npelwh80&list=PLagqLv-gqpTMx2Q10C_prJRnCtM55C0o8&index=45&t=6s), and [Introduction of Principles of DHS Sampling Weights](https://www.youtube.com/watch?v=SJRVxvdIc8s&list=PLagqLv-gqpTMx2Q10C_prJRnCtM55C0o8&index=44&t=2s). [↑](#footnote-ref-7)