Tadeusz Sozański

HANDY DATA WRITER

User's Guide

February 2000

Introduction

The purpose of this program is to help you solve the problem you will inevitably face as soon as you are done with data collection - the problem of how to move your data from a pile of paper to a computer file that must be prepared in order to carry out data analysis with the use of the **Statistical Package for the Social Sciences (SPSS).** If you can't employ an integrated package like TELEFORM to both design your questionnaire and define the list of variables whose values would be subsequently read into a database directly from completed forms by means of an optical scanner, then you will appreciate the tool I'm offering: a **user-friendly, menu-driven QBASIC 4.5 program that will enable you to produce data files for use with SPSS**, You will be happy with the result, although you or your typist will have to enter figures from keyboard and stare at the screen instead of feeding paper sheets to the scanner. With Handy Data Writer (HDWRITER) you can:

- set up a system of 1 to 4 databases appropriately structured to store in them the values of up to 600 variables;
- fill out the databases by writing successive records on the screen data form with default or customized layout;
- view and edit already written records;
- append records from a database having the same structure (say, filled out by your collaborator using the same program);
- delete marked records from a database;
- sort a database with respect to 'record identification mark';
- create a text file which can be printed on paper to obtain code sheets compatible with the screen layout;
- rewrite the QBASIC database into a file which can be read into SPSS by the DATA LIST command;
- automatically write the file containing the DATA LIST command in question.

Detailed instructions on how to operate the program are given later in this guide. You should read it prior to editing your codebook in order to learn what information on variables has to be passed to HDWRITER in a **Variable Definition File**, an input file that the program must read at the outset. If you accept the defaults, you will have to write only a few lines to produce this file. To help you edit VAR files, file BLANK.VAR is attached with the program. It is a blank form you should fill out as soon

as you have edited your **codebook**. Use EXAMPLE.VAR to test the program before you start work with your own variable definition file.

Installation, getting in and out of the program.

HDWRITER was written to work under DOS. If you have a switch in your AUTOEXEC.BAT between WINDOWS 95/ 98 and subordinate DOS, select DOS. If you don't have this option, switch to the MS DOS Mode. I did not test the program under later versions of Windows. I hope it works, too.

No installation is needed, just copy HDWRITER.EXE and your VAR file to a directory that will stay your **work directory** all the time. All files read and written by the program will also reside in it.

When you load the program and see the first screen with my email address (feel free to send any comments), press Enter to start work or Esc to exit. When you choose to run the program, you will be prompted at the very beginning to enter the name of a Variable Definition File. Type the filename, press Enter, and wait until the VAR file is read. If the file is not found in your work directory, the program will let you know about it with the message 'File not found'. For any other error, the error trapping procedure returns only the error number. I included this procedure in the program to locate possible errors, though a scrutiny of the source text and a number of tests make me hope that the program does not have a serious flaw.

The data read from a VAR file is checked for consistency with the program requirements. Check your VAR file, if the execution stops before the **Database Selection Menu** is displayed on the screen. Use this menu to get out of the program or to pick out a database to work with. When you choose a database you will see the first part of the **Operations Menu** (operations A-D). Press Down cursor to see the remaining operations (E-I) available for the database.

You can't stop the execution of the program whenever you like to. If you want to get out of the program, complete the current operation or wait until the program's action comes to an end (only the sorting operation may probably take some time, if your computer is slow). Then go back to Database Selection Menu and press Esc.

Databases, records, and variables.

The term **database** is referred in this manual to any text file that is structured within QBASIC as an ordered sequence of **records** of fixed **length**. Records of length *m* are ordered sequences of *m* characters. The elements of a record of length *m* are said to be located in *m* numbered **columns**. Thus, the total number of characters in a file which consists of *n* records of length *m* is *nm*.

Within a QBASIC program, records are identified by their **record reference numbers**. You can write a new record and append at the end of the database, read an existing record and extract the information you need from a given column, you can also modify the record and put it back to the database. In QBASIC, databases are called **random access files**, because the records can be accessed 'randomly', that is, in any order, not necessarily in the order of their location in the database. Outside QBASIC such a file is just a long string of characters without an end-of-file mark at the end. The size in bytes of such a text file is exactly the number of characters in it. A structure is introduced to a random access file within a QBASIC program: a 100 byte file can be read as a sequence of 10 records of length 10 or a sequence of 20 records of length 5, etc.

The databases created and processed by HDWRITER are recognized by their common filename extension (**filetype**) RAF. The **database names** must be provided by the user in his Variable Definition File. These names are also used by the program as filenames with other filetypes (DAT, DLT, FRM) to create files with different content and structure. We explain all filetypes in the next

section. For now, let us say only that whenever you are prompted by the program to enter a **filename**, don't type the extension. More exactly, you will have only two opportunities to enter a filename while running HDWRITER: at the very beginning when you are asked to specify the name of your Variable Definition File (it must have filetype VAR), and when you need to point out a file from which you want to append records to your currently processed database.

The VAR file and the respective RAF files are your most important **source files**. The VAR file is a formal representation of your codebook, while the RAF files are devised to store the data transferred from your filled-out code sheets. Make backup copies of the RAF files during and at the end of the data typing process in order not to lose the results of your work. Other files, in particular, the ASCII text files you will need to put the data into SPSS, can be produced from the source files anytime you run HDWRITER.

The RAF files can be used in other QBASIC programs. For example, it is not difficult to write a program that will select a random sample of records from a given database. You will need then to know the record length of the database. The names of the databases, the ranges of variables stored in them, and the respective records lengths are displayed on the screen as soon as the Variable Definition File is read by the program.

Records correspond to the **units of analysis**, usually called **cases** (respondents, questionnaires). The cases in one sample are characterized by the same set of **variables**. The *i*th record in a given database contains values of the variables assumed by the *i*th unit of analysis. Unfortunately, sociologists hardly ever respect the Ockham rule, they like to define not tens but hundreds of variables. As a consequence, one must often split the set of variables into a few successive **segments**, and place each segment in a separate database. We say 'segment' rather than 'subset' because HDWRITER requires that the variable set have the form of a **list** of numbered items. For example, if you have 300 variables, you can place variables #1-#100 in the first database, #101–#200 in the second, and the rest in the third database. The reason for such a partitioning of the variable set is that you can't enter values of 300 variables simultaneously on one screen. The general idea I wanted to implement by designing HDWRITER is that the **layout of the screen for data entry and the graphical shape of the code sheet should be compatible and visually similar as much as possible**.

The link between two or more databases defined for a given set of cases is established by placing the same string in the **record identification field**. The field, noted ID, is common to all databases; it is located at the beginning of a record in each database. You can set the number of columns to store the identification string, or let the program to determine the length of the ID field from the planned number of records. The operation of joining partial databases into one database containing the values of all variables has not been programmed in HDWRITER. You can do it more efficiently within SPSS using JOIN MATCH command.

The ID field is followed by the sequence of fields allocated to successive variables. To define a variable you must specify the maximum number of characters that are needed to write any value of that variable. Thus, if you are 100 percent sure that your sample does not contain centenarians, assign to variable AGE a field of length 2. If you know the 'field length' of a variable and the 'field location' in the record, that is, the number of the first column in which the string of characters representing the value of the variable begins, you know precisely where to find the value of the variable. Equivalently, you may assign to a variable the successive column numbers, say, 12-14, which means that the field length is 3, and the value of the example variable is written in columns starting from 12th.

However, the **field length** is a more important characteristic of a variable than its **location in a record**. First, the variable field length does not depend on the position of the variable on the list and hence it can be 'locally' changed (say, if you unexpectedly find a centenarian in your sample) without having to rewrite the whole codebook. Secondly, it determines the range of values the variable can assume. Thus, when you are going to use HDWRITER, do not assign columns to the variables in your codebook. Do not follow the researchers who are used to do that since they learnt the techniques of data handling when the punched cards were in common use. Instead, prepare the numbered list of

variables and attach the field lengths to each variable. You can also specify for some or all variables the **minimum value** and the **maximum value** that a given variable may assume. If you pass this information to HDWRITER the program will prompt you to correct the mistyped value. If, for example, you've typed 2 while the declared range of values is 0 to 1, the error will be communicated immediately and you will be asked to correct or ignore it. Do not reserve a special symbol for missing data point, but leave the field blank (filled with spaces) to allow SPSS to translate this to 'system missing value' (see the last section for more details).

To write a database with HDWRITER you need only **variable reference numbers** and field lengths, the location of a field is determined automatically when you tell the program to generate the DATA LIST command file. However, the SPSS command in question does not make reference to the numbers of the variables to but to their symbolic names. The program produces default names for variables from their numbers: if you have defined 100 variables, their names will be V001, V002,...,V100. I strongly recommend to invent one's own names that will override the default names. When you define **variables names**, you must keep in mind the SPSS rules: the variables must have distinct names, each name must be at most 8 characters long and begin from a letter. Besides letters and decimal figures HDWRITER allows you to make use of the character '_'; use it to produce names of the form Q1_1,.., Q1_10 (question #1, items #1-#10). Work out a system of naming variables as early as possible. It is best to do it at the same time when you are editing the questionnaire. See the last section for more tips on codebook construction.

The identification string is passed to SPSS by the DATA LIST command along with the values of other variables. ID is treated by SPSS like other variables, however, it is the only **text variable** whose values are strings of ASCII characters. All other variables are declared to SPSS as **numeric variables**. HDWRITER requires that all variables defined in a VAR file take only **nonnegative integers** as values. Since the **maximum variable field length** accepted by HDWRITER is 4, the range of values primary variables can assume is 0-9999. If you must deal with a variable which takes both positive and negative values, define an auxiliary variable to code the sign of a value. Let S and A stand for the sign and absolute value of an integer-valued variable X. If you code the plus sign with 0 and the minus sign with 1, then the command COMPUTE X=(1-2*S)*A will create in SPSS the variable you need. The values of a **real-valued variable** are said to have a **fixed format** if one can specify in advance the maximum number of figures before and after the decimal point, valid for all values of the variable. Such variables are not accepted by HDWRITER, but they can easily be constructed within SPSS by means of an appropriate COMPUTE command from one or two **integer-valued auxiliary source variables**.

Character set, control keys, screen, menus, files.

The HDWRITER **character set** consists of **figures** 0-9, **uppercase letters** A-Z, the **underline character _** and **space**. Lowercase letters are automatically replaced by their uppercase counterparts. All of these characters are the only characters you can use in the identification field, all but space are permitted in filenames and variable names with the restriction that the first character of a file name or variable name must always be a letter. Figures 0-9 and space are the only characters permitted in numeric fields.

The keys: **Enter**, **Esc**, **Tab**, **Up**, **Down**, **Left**, **Right**, **Bksp** are used by HDWRITER as as **control keys**. The Enter key is used to confirm the default option in a menu, to continue program execution after a pause ('Press ◂┘ to continue'), and to send a string of characters if its length is not fixed in advance (entering a filename is an example). You don't need to press Enter to enter data in fixed length fields. The Esc key is used to stop an operation and move back along the menu tree. In some circumstances the **space key** (written as [ ] in screen messages) is also used as a control key: you are prompted to press space to correct an error. Up and Down cursor keys are used to move one line up or down along the currently edited area. In some menus, these keys are assigned to options 'get previous item' or 'get next item'. When you edit a record in full screen mode, the Tab key, and Left

and Right cursor are also available (details are explained later). The backspace key allows you to delete the character just entered when you type a filename or a numeric field of length 2 or more.

At any step of the program execution, only a specified list of alphanumeric or control characters (produced by pressing control keys) is accepted as **keyboard input**. If you hit a wrong key, you will hear a beep. You can't go ahead until you press one of the right keys.

The **screen** for HDWRITER is set to 25x80 **text mode**. The lines 3-23 are used to enter values of variables, the lines 24 or 25 are reserved for messages and action prompts.

All operations are accessible from **menus**. Menu items or options are marked with letters or figures, or sometimes with symbols of other keys (Esc, cursor keys). The default option is highlighted or blinks. Options which may appear in a given menu but are not feasible in current circumstances are also displayed: their symbols are printed in square brackets. To choose an option from a menu, press a proper key; press Enter key to 'confirm' the default option.

You already know that databases and variable definition files have filetypes RAF and VAR, respectively. A variable definition file is an ASCII text file structured as a sequence of **lines**. All files created by HDWRITER except the RAF files are line-structured ASCII files. Only the lines of a DAT file may be longer than 80 characters, since each line of such a file has the same length as the record length of the RAF file with the same filename (see operation F).

A file with the filetype FRM contains a blank code sheet with cells for values of the variables to be stored in a given database. Files with filetype DLT contain DATA LIST commands (see operation E).

To sum up, HDWRITER recognizes 5 filetypes: RAF, VAR, DAT, FRM, DLT. Two operations, sorting a database and deleting records, create temporary files with extensions SRT and DEL which are replaced by the respective RAF files. You will not see the temporary files unless a power failure happens immediately before the replacement is made.

Variable Definition File.

In a VAR file the **input lines** in which the program finds the necessary information can be interspersed with **comment lines**. First four columns of each line are read first to find out if a given line contains relevant data or it should be skipped as a comment line. A line will be treated as a comment line, if it is an empty line (obtained by pressing Enter at the beginning of a new line) or its first four characters are spaces. It is recommended to print the **variable reference numbers** in these columns in all input lines which provide formal definitions of the variables. These numbers are not read by the program which counts the input lines and assigns reference numbers to variables automatically, but the numbering of lines will help you edit the VAR file. Similarly, type the **database reference numbers** in the input lines containing information on your databases. Print 0 in other input lines. You will find such markings already done in file BLANK.VAR supplied with the program. Use this file as a blank form to edit your own VAR file.

The first input line in any VAR file must contain the number of databases; it is printed in the first field (columns 6-13). The next field (columns 15-19) of the same line must contain the **target number of records**. The program will use this parameter to determine the default length of the record identification field common to all databases. If you plan to write 900 records, the identification field will be 3 columns long. If you want to increase the ID length, print the ID length in the third field of the first input line (columns 21-25). The program makes use of the target number of records also to set an upper bound for the number of records in the databases that can be processed by the program.

The first input line must be followed by the lines specifying the **database names**, followed, in the second field (columns 15-19), by the numbers of last variables in each database. Remember that no database may contain more than 8x21=168 variables. When you declare 3 databases, make sure that exactly 3 correctly filled-out input lines follow the first line. The **maximum number of databases** accepted by HDWRITER is 4. This implies that the maximum number of variables can be 4x168=672;

for consistency with SPSS/PC limitations this number is lowered to 600. The program examines the collection of parameters given in the initial lines of a VAR file, but it can't recognize and respond to all possible violations of the requirements, so take care yourself of the correctness of the top of your VAR file.

The beginning of the KGA.VAR file I prepared for my colleague is displayed below. The user KG defined 318 variables in 3 databases. He is going to enter 250 records, while his collaborator will do the rest. The latter person will use KGB.VAR file of the same form except for different database names (KG1_B, KG2_B, KG3_B) and a different planned number of records. When both complete their jobs, the program will be used to join KG1_A.RAF and KG1_B.RAF file into one database KG1.RAF (see operation I).

The next input line (marked with 0) is called **Setup Line**. The program finds here the information on which components of any variable definition are set by the user and which are to receive the default values. The setup line has 7 fields (marked f1-f7); all except the first which has 8 columns are 5 columns long.

```
   ************************************************
   |Var. Def. File  KGA.VAR  prepared by TS for KG to
   |write 250 records to databases KG1_A, KG2_A, KG3_A
   ************************************************
   | Number of databases, records, characters in ID
0 |   3    | 250 |  3  |
1 |KG1_A   | 115 | last variable
2 |KG2_A   | 213 | numbers in
3 |KG3_A   | 318 | databases 1-3
   |All setup fields f1-f7 are set by the user
   | f1     | f2  |  f3 |  f4 | f5  | f6  | f7  |
0 | Y      |  Y  |   Y |   Y |  Y  |  Y  |  Y  |
   | Definitions of variables in database 1
1 | Q1     |     |  1  |  3  |  3  | 1   | 4   |
2 | Q2ZO   |  2  |     |     |  5  |     | 72  |
3 | Q2WO   |     |     |     |     |     |     |
   (...)
```

The names of variables are stored in the first field (f1:columns 6-13). If the number of all variables is less than 100, the default variable names are: V01, V02,...,V99. If at least 100 variables are defined, then the default names are: V001, V002,... If the variable name field is left blank in the setup line, then default names are created by the program for all variables. If you want to give your own name to at least one variable, you should insert a non-space character in the variable name field of the setup line (you can type the letter Y as your answer to the question: 'Do you want to provide your own values in the given setup field'). The user's decision which is passed to the program in this manner implies that the default name will be overridden only for those variables for which the user has provided his own names in the respective input lines. You may devise names for some variables and leave this field blank for other variables, but be aware that this amounts to accepting the default names for the latter.

As soon as all input lines are read, the variable names are checked for consistency with the program

limitations. If wrong characters are used or two variables are assigned the same name, an error message appears on the screen and the execution of the program terminates.

Similar rules apply to other components of a variable definition. If all fields in the setup line are left blank, the default parameters will be generated for all variables. Then the program will not read any line past the setup line and your VAR file need not contain more than 6 input lines (the first line, the lines describing 4 databases and the setup line). However, if at least one field in the Setup Line is not left blank, you have to supply input lines for all variables and the VAR file can be pretty long. Don't worry about that. You receive with HDWRITER the BLANK.VAR file that will make your editing task fairly easy.

First, you don't have to fill in blanks everywhere. If you declare in the setup line that a given field will contain user's parameters and leave this field blank for some variables, then the default parameters will be used for these particular variables. Thus, if most of your variables take values in the range from 0 to 9, type the variable field length 2, 3, or 4 in the second field (f2:columns 15-19) of an input line only for those variables which need more than one column to code their values. Look at the portion of KGA.VAR where only variable #2 named Q2ZO is assigned non-default field length 2, while variables #1 and #3 have fields of default length 1.

The last two fields, f6 (columns 39-43) and f7 (columns 45-49), of an input line are used to tell the program the minimum and maximum value of a variable. Usually you will not need to type anything in field f6, since the default value of 0 can be accepted in most cases. The variable list coming from real research I've used here as an illustration has only few items where the minimum value is not 0. One of these cases can be seen above (variable #1, named Q1, has the minimum value of 1). Variables #1 and #2 provide examples of non-default maximum values. The default maximum values are determined by fields lengths, for variables #1 and #2 they would be 9 and 99, respectively.

The numbers given in fields f3 (columns 21-25), f4 (columns 27-31), and f5 (columns 33-37) are used by the program to determine the **layout of the screen data form** that will be filled out from keyboard. A beginner is recommended to accept the defaults or design on paper one's own screen layout and ask an advanced user of HDWRITER to translate it into appropriate numbers in the VAR file.

The sequence of all variables put into one database can be divided into subsequences called **blocks**. To assign a block to a variable one should type the **block number** in field f3 of the respective input line. The numbers given in fields f4 and f5 determine the variable location on the screen: the **column** and **row** in which the 3-figure variable number is printed. Usually all reference numbers of the variables which make up one block are located in the same column on the screen. The screen form and the code sheet generated by the program for the example database KG1_A has 7 blocks and 7 columns.

The **number of blocks** may not exceed 8. If this requirement is met, the program proceeds in turn to examine if the screen coordinates assigned to all variables make it possible for each variable to print its 3-figure number followed by colon, space and the blank field — inside the screen window in which the values of all variables in a given database are to be entered. If the screen positions assigned to particular variables overlap or are badly designed in another respect, no error message appears. To see the blank screen form and the code sheet which are generated by your particular settings, point out a given database and choose operation C from the first Operations Menu. If something is wrong or the screen layout does not answer your expectations, you have to modify the VAR file and view the result again.

If you leave blank all three fields f3, f4, and f5 in the setup line, HDWRITER will generate the entirely default screen layout. The first variable in the database will be located in row 3, column 3, blocks will coincide with the sets of variables placed in successive columns, each column will include 21 items (placed in rows 3 through 23), and the distances between columns will be automatically computed so as to take into account the maximum field length in each column. Then, if all variables in one database have the default field length 1, 8 columns each having 21 items will be obtained. Hence the maximum number of variables in a database may not exceed 168.

If you want to customize your screen layout by adjusting its shape to the particular structure of your

variable set, start from defining a partition of the variables in each database into blocks. To define blocks you should consider the structure of the variable set. The structure of the variable list in your codebook will usually reflect the **structure of the questionnaire**. Actually, a questionnaire has at least four interrelated structures: the **formal** structure (sequence of questions, nested subquestions, item lists, matrices, etc.), **logical** structure (relations between **filter questions** and subsequent **contingency questions**), **thematic** structure (partition of questions into groups concerning particular topics), and **graphic** structure (within-page layout and pagination). The graphic layout of the questionnaire may be most important for defining blocks. The coders would certainly appreciate coding in a separate block the stuff from a single page of your questionnaire. When you write or edit a record in the full screen mode, you can skip a block (that is, leave fields blank in there) and jump to the next block. Then, it may be convenient to put into one block the series of items which are contingent on a particular answer to a filter question. Automatic skipping or completing fields for contingency questions has not been programmed in HDWRITER.

```
      _ _ _
   ID:[_|_|_]                                                  Database  KG1
-------------------------------------------------------------------------------
|                                                                             |
|      _          _          _ _        _ _        _          _ _         _   |
|   1:[_]      19:[_]     37:[_|_]   58:[_|_]   75:[_]_    94:[_|_]  112:[_]   |
|      _ _     20:[_]     38:[_]_    59:[_]_    76:[_|_]   95:[_|_]  113:[_]   |
|   2:[_|_]    21:[_]     39:[_|_]   60:[_|_]   77:[_]     96:[_|_]  114:[_]   |
|   3:[_]_     22:[_]     40:[_]_    61:[_]_    78:[_]     97:[_|_]  115:[_]   |
|   4:[_|_]    23:[_]     41:[_|_]   62:[_|_]       _ _    98:[_|_]            |
|   5:[_]      24:[_]     42:[_]_    63:[_]_    79:[_|_]   99:[_|_]            |
|      _       25:[_]     43:[_|_]   64:[_|_]   80:[_|_]  100:[_|_]            |
|   6:[_]      26:[_]     44:[_|_]        _     81:[_|_]  101:[_|_]            |
|   7:[_]_          _     45:[_]_    65:[_]     82:[_|_]        _              |
|   8:[_|_]    27:[_]     46:[_|_]        _     83:[_|_]  102:[_]              |
|      _ _     28:[_]     47:[_]_    66:[_]     84:[_|_]  103:[_]              |
|   9:[_|_]    29:[_]     48:[_|_]   67:[_]     85:[_|_]        _              |
|  10:[_|_]_        _ _   49:[_]_    68:[_]     86:[_|_]  104:[_]              |
|  11:[_|_|_] 30:[_|_]    50:[_|_]   69:[_]     87:[_|_]  105:[_]              |
|  12:[_]_     31:[_]_    51:[_|_]   70:[_]     88:[_|_]  106:[_]              |
|  13:[_|_]    32:[_|_]   52:[_]_         _     89:[_|_]  107:[_]              |
|  14:[_|_]    33:[_]_    53:[_|_]   71:[_]     90:[_|_]  108:[_]              |
|  15:[_]_     34:[_|_]   54:[_]_    72:[_]     91:[_|_]  109:[_]              |
|  16:[_|_]    35:[_]_    55:[_|_]   73:[_]_ _  92:[_|_]  110:[_]              |
|  17:[_|_]    36:[_|_]   56:[_]_    74:[_|_|_] 93:[_|_]  111:[_]              |
|  18:[_]                 57:[_|_]                                            |
-------------------------------------------------------------------------------
```

Once you have partitioned your set of variables into blocks, you can tell the program to determine the default column for the variables in each block. Give your own non-default column parameter in field f4, only if the default system of columns appears too loose or too tight.

In most cases, the default, unbroken sequence of rows within one column is OK. Use a non-default setting for the row parameter (field f5) to divide blocks into **subblocks** (to make possible jumping between them while editing a record). Subblocks are groups of variables within a block that are separated on the screen by one or more empty rows or lie in different columns. The example is given in the code sheet for KG1 where the first block placed all in column 3 consists of 4 subblocks.

Even if you have set up the fields f3, f4 or f5 so as to make the program read user's parameters, you don't need to write numbers in every cell. When you edit your Variable Definition File have in mind the following rules.

If the block field (f3) is found blank for the variable currently read from the VAR file, then the block number of the previous variable is assigned to the current variable.

If the row field (f5) is found blank for the current variable, then the next row is assigned to it where 'next row' means the row attached to the previous variable plus 1, or row 3, if row 23 has been assigned to the latter.

If the column field (f4) is found blank for the current variable, then the program compares the rows assigned to the current and previous variables. If Row(i)>Row(i-1) then Col(i)=Col(i-1); otherwise the 'next available column' is computed.

These rules are part of the algorithm which determines recursively the block, column, and row numbers for any variable for all combinations of settings in fields f3, f4, and f5. The full description of this algorithm would be too long and complicated to include in this guide. You can see how the algorithm works by changing settings in the EXAMPLE.VAR file which is supplied with HDWRITER.

Once you've learnt the basics I aimed to explain as clearly as possible, you should be able to map your codebook onto the Variable Definition File. First, make a copy of the BLANK.VAR file. Give the copy a different name to save the template for future use. Next, edit the copy by means of an ASCII text file editor. If you use an advanced wordprocessor like WordPerfect or Word, remember to save the result of your editing in the ASCII text format.

Database operations

When you point out a database to process, its name and the number of records are displayed in the upper window of the screen. Since then the database you've picked out becomes your **Currently Processed Database** and all operations can affect only it. The operations marked with letters A-I are listed in the lower window in two **Operations Menus**; use Up and Down cursor to switch between the two menus. To change the current database you have to leave the first Operations Menu for the Database Selection Menu.

A. Writing new records to the database.

This option is available only if the number of already written records is less than the target number of records declared in the VAR file. When you press A, you will be asked first to choose between two **screen modes of data entry**. Next you are prompted to tell the program whether you want it to automatically generate the identification number for any record or you would like to enter it yourself from keyboard. Let us first discuss these two options of which the latter, used more often, is the default option.

Record Identification Field.

Program generation of the record identification string consists in the automatic rewriting of the record number as a string of figures with leading 0s. If the ID field is 3 columns long, then record #1 will receive the identification string '001', etc. I recommend to choose this option only if you've got one database and you need ID only to count and distinguish cases. If so, bring the pile of code sheets (you don't have to write down IDs on them in advance) to your desk and rewrite them one by one to the file in any order. When you have entered the data from a given sheet, write the record number on that sheet and get the next one. If numeric IDs are already written on the sheets (you may tell your

coders to do that), you will have to sort the sheets in ascending order of numbers to ensure that automatically generated IDs and those written on the sheets perfectly match.

If you have more than one database, write down ID on every questionnaire and instruct the coders to rewrite it to all code sheets. Clearly, for any case the same ID should be written on the respective code sheets corresponding to particular databases. Count all cases and determine the length of the ID field accordingly. HDWRITER requires that the ID field have at most 5 columns an no less than are needed to assign different identification strings to all cases using the natural **system of successive fixed length figure strings**. That is, if the number of cases *n* lies within the range from 100 to 999, then these strings will take the form 001, 002,...,###, where ### stands for the 3-digit string representing *n*. Decimal figures are not the only characters that can be typed in the ID field. The letters A-Z, the underline character and space are also permitted. A record for which the ID field is blank (covered with spaces) is interpreted by HDWRITER as **marked for deletion**.

The use of string representations of numbers (with leading 0s or spaces) involves a natural one-to-one correspondence between the set of record identification strings and record numbers. Moreover, when you enter records in the order determined by ascending IDs, the identification number, typed at the left end of the top screen line, will always match the number of the current record displayed at the right end. If you find such an agreement desirable, you need to order the collection of all code sheets before you start rewriting them to the database. However, you may enter all records in any order and next sort the database by record ID (see operation G).

The system of identification numbers I've described above is most common. However, there are at least three situations where identification marks may be constructed on a different basis (then they would not match record numbers in the sorted database). First, you may code in the ID field the information that you consider important, but wouldn't like to code it by means of a regular variable characterizing cases (e.g. one can store in the ID field the initials of an interviewer followed by the number of a particular interview made by him or her). Secondly, if the questionnaires mailed to all potential respondents are supplied with serial numbers, you may use them as identification numbers for the cases which remain your effective sample (the questionnaires which were returned completed). Thirdly, when you repeat the research on the same sample, you have to assign to the subjects the identification marks given to them in the first survey.

As soon as you've typed the identification string for the current record the program examines if it has already been used for another record. Repetitions in the ID field are permitted only if the string contains spaces alone. If a duplicate is found for a non-all-spaced string, you will be prompted to correct or ignore the error. If no error is found, you still have an opportunity to retype ID (press space) or accept it (press Enter).

Entering values of variables in two screen modes

Once the identification string has been entered and accepted, the arrow which points to the **Currently Edited Field** goes down to the **Variable Values Entry Window** that will contain variable numbers followed by blank fields for all variables in the current database. As you already know, the layout of cells in this rectangle depends on the parameters given in your Variable Definition File.

You can fill out such a form in either of two modes of data entry: the **sequential write-down mode** or the **full screen edit mode**. With sequential mode selected, when you enter the window, you will see in there only the first block of variables. When you finish typing the values of all variables in the block, the next variable block will pop up in the window and so on until you type the value of the last variable in the database. All fields must be completed sequentially. You can't move on to the next field until you have typed the data into the current field. However, you can always move up or down the whole area extending from the first variable field to the current field. Use Up and Down cursor to move throughout this area. Remember that the cursor keys (and other control keys available in the full screen mode) work only if you are about to edit a given field, but have not yet entered a character in it.

If a field has only one column, the arrow shifts down to the next field as soon as you press space or 0-9. Entering data in a **multi-column field** is subject to the following rules: (i) When you entered a character in the first column, you must enter characters in remaining columns to get out of the field; (ii) When you are in the middle of the field, you can press the backspace key to step back and delete the character just entered. When you reach the beginning of the field, you can't get out of it to the previous field, but you have to retype the whole string and pass to the next field where the cursor works again; (iii) Leading spaces are permitted (you can enter value 1 in a two-column field as 01 or [space]1), but you will hear a beep, if you press space after a figure.

Once you have entered data in the current field, HDWRITER checks if the value of the current variable fits the range declared in the Variable Definition File. If you mistyped the value, you are prompted to correct the error (press space) or ignore it (press Esc).

The principles of data entry in a single field remain in force in the full screen mode. The difference between the two modes lies, first, in that in the full screen edit mode you receive at once the access to all variables in the current database (the whole form appears on the screen) and you can move down regardless of whether the fields passed by are already filled with numbers or remain blank. Secondly, you can jump between blocks and subblocks. Press Right (Left) cursor to move the arrow to the next (previous) block. The Tab key allows you to pass to the next subblock. Lastly, you can finish editing a record at any moment by pressing the Esc key (recall that the cursor keys, the Tab key and Esc key are inactive when you are inside a multi-column field). When you leave the Variable Values Entry Window the skipped fields will be filled with spaces.

The full screen mode is automatically set when you answer 'No' to the question 'OK?' which appears at the bottom of the screen when you finished editing the current record. Type N, if you would like to correct just entered record or Y if you want the record to be appended to the database. Since 'Yes' is the default option, press Enter in order to leave the **Record Editing Screen**. The menu displayed on the next screen allows you to write the next record (press Down cursor) or to stop the operation of writing new records (press Esc key).

If you choose to stop work, HDWRITER will inform you about the **number of records entered during the session** just finished and will let you know the average time (in minutes and decimal fractions of a minute) of writing one record. Before you start regular work, enter a few records and multiply the **mean writing time per record** by the target number of records to estimate the total time needed to rewrite all code sheets into the database. If you hire a typist, you will know how long he or she will work, so that you can pay your worker for the job accordingly.

B. Displaying and editing already written records.

When you select operation B (available only if the current database has at least one record) from the first Operations Menu, the program will offer you two options of record retrieval: **search by record number** (option 1) or **search by record identification string** (option 2). If you choose the second option, you will be prompted to enter ID of the record you want to view. If such a record exists, it will be displayed on the screen. If you want to modify the record, reply Y(es) to the question 'Edit?' which appears in the bottom line. You will then be given first the opportunity to change the record ID (in particular, to mark the record for deletion) and next to edit variable values in the full screen mode. Use control keys to quickly get to the fields which need correction and press Esc to quickly exit the Variable Values Entry Window when you are done.

If you reply N(o) to the 'Edit?' question (it's the default answer), you will return to the **Record Search Menu** in which the two search methods are given along with the option to finish the database inspection (Esc). Since all records which are not marked for deletion are supposed to have unique identification marks (it is but a suggestion which can be ignored), the search of the next record with the same ID has not been programmed: if you enter the same string for the second time, you will see the same record, the first one with the given ID in the database. If you want to look through all records

marked for deletion, you should first sort the database by the ID field. You will find the records with all-spaced ID at the beginning of the sorted database. To view them one by one, tell the program to search records by record number.

If you select search by record number, you are prompted first to type the record number in the range from 1 to *n* where *n* is the number of records in the current database. If you type a number beyond the range, you'll have to re-enter it. If you press Enter at the prompt, the last (*n*th) record will be displayed. After the record with the entered number, you can see the next or previous record by pressing Down or Up cursor, respectively. Thus, you can start from the first or the last record and view all records in the current database, getting them one by one in the order of decreasing or increasing reference numbers. If you want to check if your typist does not make many errors, browse through already written records and compare what you see on the screen with the data on the code sheets.

C. Displaying screen data form and generating code sheets.

It is the operation you should call first as soon as the program reads your Variable Definition File for the first time. If you press C for the current database, you will see the screen data form you have designed. Next press Enter to see the code sheet and decide whether to save it in a file. To obtain a hard copy of a code sheet, you can print such a file but you have to use a typeface of fixed pitch (12 characters per inch is the best option). The code sheet, size of a half of a standard A4 page, can be enlarged by means of a copier. Do it, if your coders prefer larger boxes for writing numbers.

The file named after the database with extension FRM added by the program is written to disk anew any time when you run the program and answer Y(es) to the question of whether to save the code sheet. If you want to keep the earlier version of the code sheet, generated under old settings you are going to modify, change the name of the already existing FRM file before you tell the program to produce the new one.

D. Displaying variable definitions.

When you call this operation, you will see the variable names displayed in the window in the same positions where the variable numbers are printed inside the screen data form. One column left to a variable name the field length is shown, provided that it is greater than the default value of 1. Press Enter to see in turn the minimum values for all variables in the current database. The non-default, non-zero values are highlighted to mark that non-default settings are seldom used for this parameter. The next screen shows the maximum values. Now the default maximum values are highlighted, since for this parameter the non-default setting is a rule.

E. Generating DATA LIST command file.

The DATA LIST file which has extension DLT is written automatically whenever operation E (first in the second Operations Menu) is called for the current database. The first line of a DLT file has the form DATA LIST FILE='filename.DAT' FIXED /ID 1-# (A); the remaining lines contain the names of the variables followed by column ranges. Use INCLUDE 'filename.DLT' command to read the data into SPSS from the DAT file.

An already existing DLT file is always overwritten, so if you want to keep the old version, you must rename it or move to another directory. The database itself need not be ready to produce the DLT file as the information needed to write it is taken solely from the Variable Definition File.

F. Producing the SPSS fixed format data files.

A database which has been produced by HDWRITER as a QBASIC random access file can be further manipulated within the program (see operations G, H, and I), but it can't be used as input file in SPSS. However, you can easily copy any non-empty database to a **fixed line-length ASCII file** with the same name and extension changed to DAT. The DAT file is a little larger than the source RAF file, as it is produced by adding 'carriage return - line feed' characters at the end of each record. When operation F is called for the current database, the program checks if the DAT file for this database already exists. If the file is not found, it is created (you will see the message and hear the hard disk work). If a DAT file with the given name already exists, the program will ask you if you want to overwrite it or abandon the operation.

G. Sorting a database with respect to the ID field.

If the variables characterizing the common set of cases are allocated to more than one database, and the records in each database have been entered in a different order, you have to sort the databases with respect to record ID prior to producing fixed format data files. The DAT files must be read separately into SPSS and saved in separate system files. If you want to use the JOIN MATCH command to integrate these files into one system file containing all variables, you have to make sure that the cases in each file match each other. This condition will be met, if the records in each source database are identically ordered with respect to the ID field which is common to all databases and uniquely determines each case.

The sorting operation rearranges records in a database in such a way that record IDs in the sorted database form an ascending sequence with respect to the relation of lexicographic order on the set of fixed length ASCII strings. The latter relation is determined in turn by the standard ordering of ASCII characters according to which space precedes decimal figures 0-9 which precede uppercase letters A-Z and the underline character _.

When operation G is called, the program checks first if the current database is already sorted. If it is not sorted, the user is asked to confirm his intention to carry out sorting. The test I made showed that the computer equipped with the processor Intel Pentium 100 Mhz needed some 7 minutes to sort and rewrite the database with 5000 records. When the sorted set of records is rewritten to a temporary file, the original file is erased and the new file is renamed to receive the old name. Thus, if you want to store the original, unsorted file, make its copy with a different name before sorting.

H. Deleting records from a database

If you want to delete some cases (e.g. those with lots of missing data points) from your sample prior to transferring the whole data set to SPSS, you can carry out the delete operation with HDWRITER. Locate a record you want to delete (operation B) and enter spaces in its identification field. Don't forget to do it for the records with same ID in all other databases defined in your Variable Definition File.

When all records you want to delete are marked, call operation H separately for each database. The program finds first the number of records in the current database which are marked for deletion. If there are any such records, you will be asked to confirm or abort deletion. If you choose to delete records from the current database, the records which are not marked for deletion will be rewritten to a temporary file. Next the original file is erased and the temporary file is renamed appropriately. Thus, if you want to store the original file, you should make its copy with a different name prior to deleting

records.

I. Appending records from another database

When you order the append operation, you will be prompted to type the name of a database from which records are to be appended to the current database. The program opens the RAF file with the name you've entered and verifies whether it is not empty and whether its size in bytes is a multiple of the record length of the current database. If the latter formal condition is met, the number of records in the source database is computed and displayed in the bottom screen line. The yes-or-no question which appears next allows the user to abandon or confirm the intention to append records.

If you employ two or more typists, prepare a separate variable definition file for each of them by modifying the parameters given in the initial lines of your master VAR file. Assign to each typist a proper target number of records and give distinct names to the databases they are to complete. When the partial databases are ready, append them sequentially to your master database which may be empty before receiving records from the first source database. Note that the database obtained in such a way may need to be sorted.

How to prepare a codebook for use with HDWRITER and SPSS

To define a **variable** one needs to show how a **numeric value** is assigned to any **case**. Two types of variables, primary and secondary variables, can be distinguished with respect to the way in which a variable is operationally defined. **Secondary variables** are constructed from already defined variables by applying to them various formal operations like 'recoding' or arithmetic operations. For example, one can produce a new variable with a smaller number of values by lumping together certain values of a given variable. One can also add up two variables to obtain the third one, etc.

In the social sciences, **primary variables** are usually obtained by 'coding' answers or responses of a set of subjects to a number of closed-ended or open-ended questions or tasks. The tool one needs to transform the **source material** (completed questionnaires) into the **data matrix** (the table in which the *ij* entry is the value of *j*th variable for *i*th case) is called the **codebook**. On the one hand, the codebook contains **coding instructions** that are passed to the coders whose task is to assign values of all variables to all cases by processing questionnaire records according to the prescriptions supplied by the researcher. On the other hand, the codebook must provide formal descriptions of all variables that will be used by SPSS and the data entry program. The formal part of the codebook is more important for your consultant on data analysis than for the coder. The coder expects from the researcher clear, simple, and workable instructions that allow him to easily and unequivocally classify every case into a category marked with a number. The coder does not need to know what use will be made of the data he has produced. The data analyst will, in turn, be happy, if he finds in the codebook all formal elements of a variable definition required by SPSS. He will certainly ask the researcher to provide symbolic names of variables and their value ranges, but will be little interested in the procedure by which variable values are assigned to cases.

Thus, to construct a good, workable codebook the researcher should have in mind its double function and the interests of the two users of this tool. There is one matter where the competence of the analyst may help the coder. The codebook should contain, first of all, the definitions of primary variables. If you want to include a secondary variable in your codebook, first ask your consultant if the variable in question can be constructed by means of SPSS commands. If the procedure defining a secondary variable is complex and requires from the coders to analyze configurations of values of many primary variables, try to spare them the work that can be done by a computer program. However, in some circumstances you may find more convenient to entrust to the coders the task that could be done within SPSS but would imply writing complex commands to produce new variables. Such a situation happens, for example, when auxiliary variables are used to code responses to a question with a 'conjunctive cafeteria'. A **cafeteria** is the list of items or options attached with a

closed-ended question. If the respondent is instructed to pick out exactly one answer, the cafeteria is said to be **disjunctive**; if one is allowed to select more than one item from the list, the cafeteria is said to be **conjunctive**. If the list is long, e.g. it has 10 items, and the respondents are forced to mark at most, say, 3 options, or the inspection of the completed questionnaires reveals that the number of marked items has never exceeded 3, then subjects' choices can be coded by means of 3 auxiliary variables. The value of the first variable is the number of the first item selected by the respondent; if it is the only marked item, the two other variables receive the value 0, etc. Such a coding method was very popular when punched cards were in common use, because one could economize on the number of columns by filling 3 instead of 10 columns needed to code the same information by means of a sequence of zero-one variables. However, if you save on columns, you will have to create the zero-one variables later within SPSS (by means of IF command), because the auxiliary variables are pseudo-variables; in particular, you can't crosstabulate these variables with regular variables. As a consequence, I have always suggested to apply zero-one method to code questions with conjunctive cafeteria, even if this would imply an increase of the number of variables. Now it is no longer a problem: HDWRITER can cope with large variable sets, and the coders should not protest, either, since the zero-one coding method is not much more time consuming.

If you are going to use HDWRITER to produce your data file and carry out the data analysis by means of SPSS you should give the appropriate field structure to each item in your codebook. Begin writing the codebook from setting the format of the identification string, but **do not include ID in the list of variables**. The recommended description of a variable looks as follows.

1. The variable reference number;
2. The variable name;
3. Field length;
4. Range of values (minimum and maximum value);
5. The way of coding missing data points;
6. Variable description string and value description strings;
7. Operational definition of the variable;
8. Processing directions.

The codebook edited in such a way is the **list of definitions of numbered variables**. Its format much differs from the traditional format (list of questions copied from the questionnaire, supplied each with coding prescriptions and column range, usually printed in the last field). You should think in terms of variables not questions in order to be able to design statistical analyses in a way that will be understandable to your consultant.

If you have one team of coders to code the whole questionnaire, adjust the order of variables to the structure of the questionnaire to make easier the coding process. The quality of the codebook much depends on how the questionnaire is structured. Prepare the first draft of your codebook when you are editing your questionnaire in order not to discover afterwards that letters were more appropriate to mark some items than numbers, or conversely.

The order of variables in the codebook will be transferred to HDWRITER and SPSS (you can change it within SPSS). Next to the variable number, put the symbolic name of the variable, its field length, and the lowest and highest value. The information for each variable given in the codebook fields #1–#4 is needed to edit the input file for HDWRITER.

The fields #5 and #6 contain data needed only by SPSS. SPSS distinguishes between **system missing value** and **user missing value**. A system missing value is assigned to a case for a given variable by the DATA LIST command, if the variable field in the line representing the case in the DAT file is filled all with spaces. To be treated as user missing value, a given numeric value must be

declared as such in MISSING VALUE command. You don't have to declare user missing values for variables obtained by coding answers to questions which are logically unrelated to other questions. Thus, the values of any two-valued variable defined on the whole sample can be coded as 0 and 1 in line with the statistical standard for coding dichotomous variables.

My old dBASE III+ programs were unable to distinguish between 0 and space. As a consequences I recommended the users to code 'no answer' as 0. I also suggested to code 'not apply' category by a number higher by 1 than the number assigned to the last 'proper' value of a variable. These directions are no longer valid in HDWRITER. Use 0 as a standard user missing value for all variables listed in MISSING VALUE command, but do not introduce user missing values where the system missing value would be more appropriate. Use the system missing value (that is, leave the variable field spaced) to code the 'not apply' category in contingency questions. You will then be able to skip 'not apply' fields while editing records in full screen mode. Remember that a variable obtained by coding a contingency question is defined only for those cases for which the respective filter variable took a given value or range of values. For such a variable, you can define also the user missing value, and then let it be 0 for the sake of uniformity.

You should also supply most variables with short descriptions that roughly inform what is the meaning of a given variable or what source data was used to construct it. These text strings, called **variable labels**, are optional in SPSS (don't confuse the variable label with the variable name!) and not needed by HDWRITER. However, defining variable labels can help you to design statistical analyses and decipher the output files or printouts, especially if the variable list is long. The VARIABLE LABEL command usually follows the MISSING VALUE, and DATA LIST command in the process of preparing the SPSS system data file that will be the base of all analyses. I recommend to define variable descriptions well in advance and include them in the codebook, to have them ready to put into SPSS, when you discover that dressed variables look better than bare. Variable descriptions are useful also for the coders, they help them to learn the associations between variables and questions.

Do not abuse the SPSS option of attaching **value labels** to variable values. If the symbolic name of a variable is meaningful by itself, put the information on the values into the variable label rather than into value labels. Thus, if you call SEX the variable with values 1 and 0 and attach with it the label '1=M; 0=F', you will know everything about this variable.

Field #7 of the codebook which contains coding instructions is called here the **operational definition of a variable**. This part of the codebook is of prime importance for the coders who receive here the rules and directions on how to assign a value of a given variable to any case. The minimal information you can put in this field is the reference number or symbol of an item or question in your questionnaire, followed by the point 'Precoded'. A closed-ended question is said to be **precoded** if the value of the variable associated with the question can be found for any case in the completed questionnaire. Then the coder's task is only to rewrite to the proper field in the code sheet the value found in a given place of the questionnaire. Unfortunately, few researchers construct questionnaires so as to minimize their future work on the codebook and to spare the coders unnecessary work.

The coding prescriptions for open-ended questions must of course be more elaborate and can't be prepared simultaneously with the questionnaire. You can't choose an adequate coding method, determine the number of categories and state their operational definitions until you get familiar with a sufficient number of recorded answers to an open-ended question. First, you have to decide on whether to construct a set of mutually exclusive categories or, instead, to identify a list of independent **themes** that can occur in each answer in various configurations. In other words, the first step in the coding of an open-ended question amounts to representing it as a closed-ended question with a disjunctive or conjunctive cafeteria constructed a posteriori through the content analysis of answers. If the coding prescription which results from the analysis is lengthy and complex, or contains a sample of typical answers rewritten from completed questionnaires, put in field #7 only the reference to the proper page in the **appendix to the codebook**.

The last field (#8) called **Processing directions** is not least important. Write down in there the information on the level of measurement of the variable, the tips as to whether and how the variable needs to be 'reduced' by lumping together some categories, etc. In general, note as early as possible

all suggestions and ideas that can help you to design statistical data analyses.

When the list of variables is ready, determine the number of databases and divide the whole set of variables into segments that will be allocated to particular databases. Next try to define variable blocks and subblocks within each database and edit the Variable Definition File accordingly. Run HDWRITER to generate code sheets. If you are not satisfied with the result, try to modify the settings in your VAR file. Finally, print the code sheets, hire the coders and assign a job to everyone. The coding of questionnaires and data entry must take some time. Use this time to start editing your list of secondary variables as a good deal of them can be constructed prior to analyzing the distributions of primary variables. The problems concerning the 'reduction of property space' and designing statistical analyses are beyond the scope of this guide.

Updated May 2004  Tadeusz Sozanski

http://www.cyf-kr.edu.pl/~ussozans/