WZB

Berlin Social Science Center

Klaudia Erhardt

NEPS–Metafile.do – a Do–File to Generate a Metafile on the Scientific Use Files of the NEPS

Application of the do-file and documentation of the resulting metafile including syntax examples on the use of the metafile

Discussion Paper

SP I 2023–501r

September 2023 (revised November 2023)

Research Area

Dynamics of Social Inequalities

Research Group

National Educational Panel Study: Vocational Training and Lifelong Learning

WZB Berlin Social Science Center
Reichpietschufer 50
10785 Berlin
Germany
www.wzb.eu

Klaudia Erhardt
NEPS–Metafile.do – a Do–File to Generate a Metafile on the Scientific Use Files of the NEPS
Application of the do-file and documentation of the resulting metafile including syntax examples on the use of the metafile

Discussion Paper SP I 2023–501r
Wissenschaftszentrum Berlin für Sozialforschung (2023)

Abstract

NEPS-Metafile.do - a Do-File to Generate a Metafile on the Scientific Use Files of the NEPS

by Klaudia Erhardt

NEPS-Metafile.do is a Stata program that generates a Stata file containing information on every variable of the NEPS Scientific Use Files. This paper documents the indicators and characteristics that are extracted from the source files and gives detailed instructions on how to apply NEPS-Metafile.do. Using the resulting metafile is demonstrated by extensive syntax examples. The complete syntax of NEPS-Metafile.do is included in the appendix and can also be downloaded (see link below).

Keywords: NEPS Scientific Use Files, metafile, data management utility, Stata syntax

last updated: 2023-11-07, text applies to do-file version NEPS-Metafile_v03-00.do

NEPS-Metafile.do

Download:	https://doi.org/10.7802/2606
NEPS-Metafile.do	Do-file to generate a metafile on the Scientific Use Files (SUF) of the National Education Panel Survey (NEPS).

System requirements:

- The do-file has been tested to run with Stata-versions 13 thru 17.
- Requires Stata IC or higher.
- The directory and file names that are to be processed by NEPS-Metafile.do must not contain blanks.

Contact to the author for questions and feedback:

erhardtk@gmx.de

Contents

1. Introduction

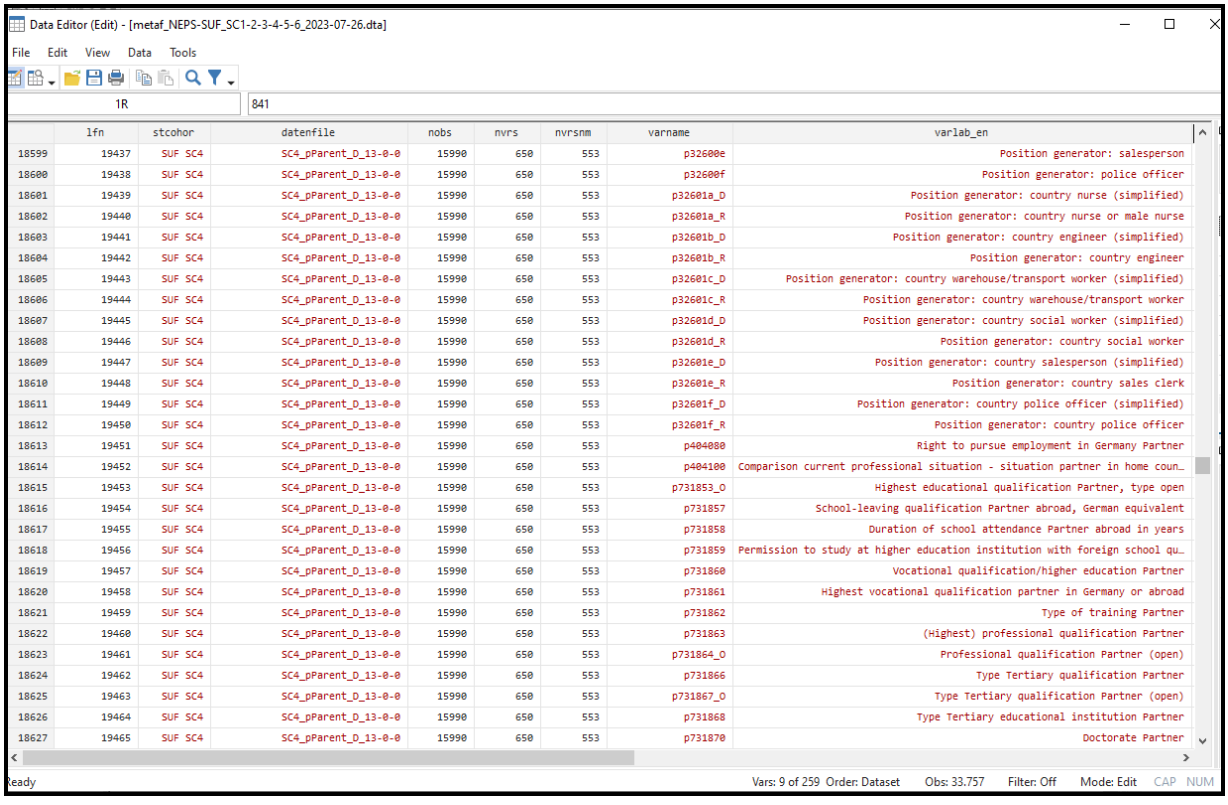
This documentation is aimed at users of the downloadable Scientific Use Files of the National Educational Panel Study (NEPS-SUF), which can be obtained from the Leibniz Institute for Educational Trajectories (LifBI) upon conclusion of a data usage agreement, see https://www.neps-data.de/Data-Center/Data-Access.

NEPS-Metafile.do is an enhanced version of the Stata program I developed, assigned by the NEPS group at the Wissenschaftszentrum Berlin (WZB), as a tool for the management of the NEPS-data. The original version was adapted to WZB-specific data storage structures and file naming conventions, whereas this new version is independent of the WZB environment.

What is the purpose of *NEPS-Metafile.do*? The program generates a Stata file containing information on all variables of every included data file of the NEPS Scientific Use File (SUF), a so-called *metafile* to the NEPS SUF files. The cases of the metafile represent the variables of the included source data files, while its variables contain the indicators that have been calculated or extracted from the original source variables.

Essentially, the metafile is a very large table containing information on the variables of the SUF-Files of the NEPS. For example, the metafile shows how many non-missing observations a source variable has, in which waves it has been surveyed, its variable name in the survey instrument.

Figure 1: View of the metafile in the data editor window of Stata



	lfn	stcohor	datenfile	nobs	nvrs	nvrsnm	varname	varlab_en
18599	19437	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32600e	Position generator: salesperson
18600	19438	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32600f	Position generator: police officer
18601	19439	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601a_D	Position generator: country nurse (simplified)
18602	19440	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601a_R	Position generator: country nurse or male nurse
18603	19441	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601b_D	Position generator: country engineer (simplified)
18604	19442	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601b_R	Position generator: country engineer
18605	19443	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601c_D	Position generator: country warehouse/transport worker (simplified)
18606	19444	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601c_R	Position generator: country warehouse/transport worker
18607	19445	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601d_D	Position generator: country social worker (simplified)
18608	19446	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601d_R	Position generator: country social worker
18609	19447	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601e_D	Position generator: country salesperson (simplified)
18610	19448	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601e_R	Position generator: country sales clerk
18611	19449	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601f_D	Position generator: country police officer (simplified)
18612	19450	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p32601f_R	Position generator: country police officer
18613	19451	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p404080	Right to pursue employment in Germany Partner
18614	19452	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p404100	Comparison current professional situation - situation partner in home coun…
18615	19453	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731853_O	Highest educational qualification Partner, type open
18616	19454	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731857	School-leaving qualification Partner abroad, German equivalent
18617	19455	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731858	Duration of school attendance Partner abroad in years
18618	19456	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731859	Permission to study at higher education institution with foreign school qu…
18619	19457	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731860	Vocational qualification/higher education Partner
18620	19458	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731861	Highest vocational qualification partner in Germany or abroad
18621	19459	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731862	Type of training Partner
18622	19460	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731863	(Highest) professional qualification Partner
18623	19461	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731864_O	Professional qualification Partner (open)
18624	19462	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731866	Type Tertiary qualification Partner
18625	19463	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731867_O	Type Tertiary qualification Partner (open)
18626	19464	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731868	Type Tertiary educational institution Partner
18627	19465	SUF SC4	SC4_pParent_D_13-0-0	15990	650	553	p731870	Doctorate Partner

To work properly, *NEPS-Metafile.do* requires the input of some parameters. In contrast to Stata ado files the required parameters are not entered while calling the procedure, but are filled in to a section in the first part of the syntax by the user. This process is explained in detail in chapter 3, while chapter 2 includes some basic explanations on the functioning of *NEPS-Metafile.do.*

Chapter 4 lists all indicators of the source data that are contained in the metafile and describes them in detail.

As said above, the metafile can be regarded as a very large synoptical table—a very simple two-dimensional data base. To use it properly, procedures using Stata syntax for data retrieval are very helpful. Chapter 5 shows by syntax examples how you can use the metafile for your tasks by extracting specific information.

NEPS-Metafile.do is not only able to process the original NEPS-SUF data but can also be used to generate a metafile for modified NEPS data. In principle, the do-file can generate metafiles for any Stata file. However, the extracted indicators are partly specific to the NEPS data, or they are based on entries in the "characteristics" of the source data files (see chapter 4.2). If those are not provided, the related variables of the metafile will contain missing values only.

2. Explanations and Definitions

2.1 Outline of how *NEPS-Metafile.do* works

The program is controlled by parameters, some of which are set automatically, while others are user defined. All parameters to be set by the user are listed in the first part of the syntax, in section "Manually Defined Parameters". Chapter 3.2 gives detailed instructions on how to do this.

Metafile_NEPS.do automatically processes the directory structure in which the NEPS SUF data files are stored. This has to be either a single source data directory that contains all NEPS data files to be processed, or a specific structure of directories and subdirectories. The directory structure of the source data files expected by *NEPS-Metafile.do* is described in detail in Section 3.1.

The source data files are processed one after the other. Certain indicators for the variables of the respective data file are extracted, temporarily stored in source data file-specific metafiles and finally merged into a new file, the overall metafile.

In order to reduce the runtime of the program, larger source data files are broken down into portions before processing (segmented processing).

NEPS-Metafile.do writes successive progress notes, potential data problems, the name and path of the resulting metafile, and the total runtime to the output and the logfile.

2.2 Definitions

Some basic terms used in this documentation are defined or explained below.

Non-missing values

Some of the indicators in the metafile relate to "non-missing" observations in the variables of the source files. The Stata missing values (., .a, .b (…) .z) and the missing codes of the NEPS Scientific Use Files (as specified in the parameter definition section of *NEPS-Metafile.do*) are excluded from the non-missing observations.

The list of missing codes can be modified by the user in the parameter definition section of the do file (see chapter 3.2).

All files of all start cohorts are treated the same with respect to the missing codes. The program does not allow for different sets of missing codes in different start cohorts or files. If missing codes are used in a file which are valid codes in other NEPS-SUF files, you will need to separately run *NEPS-Metafile.do* for those files while in each run adapting the missing codes list in the parameter definition section accordingly. The separate Metafiles can then be combined to an overall Metafile.

System-missings or sysmis

"System-missings" or "sysmis" is the denomination for the Stata missing codes ., .a, .b, (…) .z, in contrast to the NEPS-specific missing codes. The symbol for the Stata missing code "." can be easily overlooked in running text, therefore in this documentation we talk mostly of <sysmis> or *sysmis.*

Type-1 indicators and type-2 indicators

In this paper, the term "indicator" is used to describe characteristics that are extracted from the source files and their variables and transferred to the metafile.

Type-1 indicators have *one single value* per source variable. For convenience, I usually refer to them as "indicators" or "characteristics". Type-2 indicators have a *list of values* for each source variable. For instance, the maximum value of a variable is a (type-1-)indicator, while the levels of a source variable are a type-2 indicator.

A type-1 indicator loads a single metafile variable, while a type-2 indicator loads a group of metafile variables.

Macros

Within Stata syntax, macros serve as a kind of container. In the course of a Stata program, strings or values are assigned to macros to fulfil a certain task at different subsequent places in the syntax - for example as a loop counter or to transfer values to variables. Macros are crucial in *NEPS-Metafile.do* as a container for the characteristics extracted from the source files and transferred to the metafile.

There are local macros in Stata that only persist during the runtime of a Stata program, and global macros that persist throughout a whole Stata session. The complete syntax of *NEPS-Metafile.do* exclusively uses local macros. Thus no modifications of the Stata environment outside of *NEPS-Metafile.do* are made by the program.

Users of *NEPS-Metafile.do* do only encounter macros when defining the parameters to adapt the program to the user-specific environment and options.

3. How to generate a metafile on the NEPS data using *NEPS-Metafile.do*

3.1 Specification of the required directory structure to enable the automatic processing of the data of multiple start cohorts

Important note: Although Stata allows for spaces in the file names, *NEPS-Metafile.do* does not, since core functions of the do file depend on spaces being interpretable as separators between directory or file names, respectively.

As mentioned before, *NEPS-Metafile.do* automatically processes a specific directory structure in which the SUF data files are stored. This can either be a single source data directory that contains all NEPS data files to be included, or a structure of directories and subdirectories in which the data of the individual start cohorts are stored. The programming of *NEPS-Metafile.do* thus picks up the data structure in which the NEPS data is provided by LifBI, divided into (sub-)directories by start cohort and version of statistical package. Hence, the programming of *NEPS-Metafile.do* allows for the direct processing of this structure without having to reorganize the source data beforehand.

The formal rules for the directory structure that is prerequisite for a correct processing by NEPS-Metafile do are described below. How the data files of the start cohorts are distributed within this data structure is irrelevant. Hence the data files of several start cohorts can be stored in one directory, or the data files of a start cohort can be stored in different (sub-)directories. However, each source file should only exist once in any of the directories involved, otherwise it will appear more than once in the metafile.

The following rules apply:

- All data directories must be subdirectories of the same root directory.
- All data directories must be addressable with the same wildcard, for instance "SUF*". The wildcard must only address the directories of interest. For example, if you had a directory named "SUF-old-versions" in the same root directory, it would also be processed if you use the wildcard "SUF*", which is something you might not want.
- In the directories, a subdirectory may contain the source data files that are to be used for the metafile. In this case, the source data files must be in an identically named subdirectory in all SUF directories. However, if the directories hold other differently named subdirectories, these will just be ignored.

Hence, the directory structure must follow the pattern:

Rootdirectory → Data files
or
Rootdirectory → Subdirectories → Data files
or
Rootdirectory → Subdirectories → **Subsubdirectory** → Data files

Each of these levels are defined by the user in the parameter definition section of *NEPS-Metafile.do* (see chapter 3.2).

Elements in bold denominate directories that must be named positively in the parameter definition section without using a wildcard.

3.2 User defined adaptations to be set in the parameter definition section

NEPS-Metafile.do has to be adapted to the local environment and options. This is described in detail In this chapter.

The syntax of *SOEP-Metafile.do* is divided into several sections. Adaptations by the users are made exclusively in section "B) MANUALLY DEFINED PARAMETERS", which follows section "A) AUTOMATICALLY DEFINED PARAMETERS". Comments and examples inserted into the syntax, as well as the detailed specifications in the table below, will assist in determining the parameters.

Further modifications *of SOEP-Metafile.do* are not required and should only be made if you have acquired an in-depth understanding of the procedures of the do-file.

Table of the manually defined parameters

The macros listed in the table below are defined in section "B) MANUALLY DEFINED PARAMETERS". The indication "mandatory" means the macro must hold a value (i.e. the macro may not be defined as: *local xyz ""*). In contrast, optional macros may be defined as empty.

Most of the macros listed in the table are either automatically defined by a default value when not defined by the user or they are already predefined in the definition section and can be changed by the user. Therefore, the number of parameters that must actually be set by the user is considerably reduced, compared to the list.

In the table, the parameters that must be set by the user are highlighted in yellow. Alongside these, there are parameters highlighted in pale blue which must be controlled by the user and, if necessary, adapted.

The order of the macros in the table corresponds to the order in which they appear in the parameter definition section of the syntax.

In the syntax, the macros containing path and file specifications are filled with examples to be replaced by the user.

Table 1: User-Defined Parameters in the Definition Section of NEPS-Metafile.do

Name of the macro	**Explanation**
files	Criterion to select the files to be processed within the source directories. The criterion must consist of a single item with or without wildcards (i.e. "*.dta" or "*pT*.dta" or "SC4_sp*.dta")"bdp.dta" or "b*.dta" or "*p.dta"). **Optional**. If the macro is defined as empty, the default value "*.dta" is used.
rootdir	root directory to the source files **Mandatory** Note: Backslashes can cause problems with macro expansion. Therefore *NEPS-metafile.do* automatically converts all backslashes in the path information with slashes. This means that it doesn't matter whether the paths are specified here with slashes or with backslashes (applies also to the macros subd and subsubd).
subd	Criterion to select the subdirectories of *rootdir* Wildcards are permitted. **Mandatory,** if the source files are stored in subdirectories. If the source files are stored directly in the root directory, *subd* must be defined as empty.
subsubd	Criterion to select a subdirectory of *subd.* Wildcards are not permitted. The subdirectories of *subd* must be named similarly in all *subd*-directories. **Mandatory,** if the source files are stored in a subdirectory to *subd*. If the source files are stored in the root directory or in the *subd* directories, *subsubd* must be defined as empty.
pfad1d pfad1o	*pfad1d*: Target directory for the resulting metafile *pfad1o*: Target directory for the logfile. **Mandatory**. *pfad1d* and *pfad1o* may designate the same directory.
we	Appendix to the file name of the resulting metafile. Can be used to indicate the processed source files. **Optional**. If the macro is not defined or changed, the metafile of a last run at the same day with the same target directory might be overwritten inadvertedly. Predefined as "NEPS-SUF_" **Recommendation**: Control and adapt the macro definition if necessary.
we1	Additional appendix to the file name of the resulting metafile. Can be used as an indicator of the processed source files. **Optional**. Predefined as "SC1-2-3-4-5-6_" **Recommendation**: Control and adapt the macro definition, if necessary.
result	Name of the resulting metafile **Mandatory**. Modifications of the macro definition are permitted but not necessary. Predefined as "metaf_`we'`we1'`datum'". **Recommendation**: Don't change the predefinition.
reslab	Label attached to the resulting metafile using the Stata-command *label data*. **Optional**. If the macro is defined as empty the resulting metafile will not be labeled. Predefined as "metafile NEPS-SUF from NEPS-Metafile_v03-00.do"

Name of the macro	Explanation
lg	Name of the resulting logfile **Mandatory**. Modifications of the macro definition are permitted but not necessary. Predefined as "metaf_`we'`we1'`datum'.log" **Recommendation**: Don't change the predefinition.
wave1, wave2	First and last wave in the source files **Mandatory.** Adaptation of *wave2* to the latest wave in the source files may be required, while *wave1* should stay unmodified. Predefined as *wave1* = 1 and *wave2* = 18 *wave1* and *wave2* are used in *NEPS-Metafile.do* to delimit valid and invalid wave numbers. Therefore the macro definition should match the actual waves. Note: if *wave2* is defined with too small a value, the affected values of the wave indicator are designated as invalid values in the output window. No wav# variables for these values will be created. If this is the case, stop the run with "Break", correct the definition of *wave2* and restart *NEPS-Metafile.do* **Recommendation**: Don't change the predefinition of *wave1.* Control *wave2* and change it, if necessary. You can see the actual last wave from the number of the SUF releases (I.e. with SC5_D_18-0-0 the last wave is 18). If data of several start cohorts is processed, use the highest wave number in any of the included SC-specific SUFs to define *wave2*.
syrlist	Priority list of possible wave indicator variables in the source files. Note: currently the only wave indicator in the NEPS data is *wave.* **Mandatory.** Changes of the macro definition are permitted, but currently not necessary. Predefined as "wave" Note: If additional wave indicators should occur in the NEPS SUF data in future, they must be listed as a descending priority list in the *wave* macro definition. Thus *NEPS-Metafile.do* will decide which of several wave indicators present in a source file is used to generate the wave indicator variables in the metafile. **Recommendation**: Change the predefinition of *wave* only when necessary.
nepsmissundc, nepsmissdc → nepsmissings	*nepsmissings*, a combination of *nepsmissundc* and *nepsmissdc, l*ist of the NEPS missing codes. **Optional**. If the list is empty, incomplete, or contains wrong values, the "non-missing"-indicators in the metafile are not correct. *nepsmissundc* missing values found in the data, but yet undocumented. Will probably (partly) be replaced in future releases of the SUF. Predefined as "-51 -32 -19 -4" *nepsmissdc* missing values according to nepsmiss.ado from the LifBi-Nepstools, predefined as "-99/-90 -56/-52 -29/-20 -9/-5" **Recommendation**: Change the predefinition of *nepsmissing* when necessary. Probably the undocumented missing values will be documented in a new version of nepsmiss.ado and/or replaced in SUF-releases later than July, 2023.

Name of the macro	Explanation
port	Number of variables per portion when segmented processing is applied. **Optional**. If the macro is defined as empty or has the value of 0, the default value of 50 is set. This value has proved to be a good compromise between the runtime reduction by segmented processing and the runtime prolongation by the increased number of cycles over the single portions. Note: if a source file has less than twice the number of variables denominated in *port*, segmentated processing will be skipped. **Recommendation**: Don't change the predefinition of *port* which is 50.

4. Documentation of the metafile

A metafile generated with *NEPS-Metafile.do* for the NEPS SUF data contains 256 variables, i.e. characteristics of the files and variables of the SUF.

Of these 256 variables, 26 variables are type-1 indicators to the source variables, 5 variables are file-specific information and 6 variables are identifiers and technical information. The remaining 219 variables are fed from type 2 indicators: 100 variables each for the values and value labels of the source variables, and 19 variables for the up to yet 18 previous survey waves and the combination variable of the survey waves. (Explanation of type 1 and type 2 indicators see section 2.2).

4.1 The variables of a metafile generated with *NEPS-Metafile.do*

In systematic order

Identifiers

- ID of the start cohort
- Name of the source file
- Name of the source variable

File-related indicators

File-related indicators are constant over all variables of a source file.

- Data signature of the source file as date
- Number of observations in the source file
- Number of variables in the source file
- Number of variables in the source file with non-missing values
- Wave indicator: variable in the source file that holds the survey year

Variable related indicators (type-1 indicators)

Type-1 indicators have one value for each source variable.

- German variable label

- English variable label
- Variable name in the questionnaire
- Question number in the questionnaire
- German question text in the questionnaire
- English question text in the questionnaire
- Output filter in the questionnaire
- Autofill instruction in the questionnaire
- Number of non-missing observations
- Number of system-missings
- Number of levels
- Number of non-missing levels
- Value of the variable if there is only 1 level
- Value of the variable if there is only 1 non-missing level
- Frequency of value 0
- Minimal value
- Minimal non-missing value
- Maximal value (except system-missings)
- Name of the value label definition
- Number of unlabeled observations (only with labeled variables)
- Highest labeled value
- Minimum length (string variable)
- Maximum length (string variable)
- Number of survey waves
- Earliest survey waver
- Latest survey wave

Variable related type-2 indicators

Type-2 indicators have a series of values for each source variable and feed a related series of metafile variables.

- Survey waves
- Combination variable of the survey waves
- Levels (variables with max. 100 levels)
- Value labels (variables with max. 100 levels)

"Technical" variables

- Consecutive number
- Data type of the source variable
- Storage type of the source variable

4.2 Detailed documentation of the metafile-variables

Table 2: The metafile variables in the order in which they appear in the metafile

Variable	Variable label	Description
lfn	Record Number	numVar, range: Integers > 0 Consecutive number of all observations; represents the sequence of the variables of the source files at the time when the metafile was generated. *lfn* is generated upon each run of *NEPS-Metafile.do* and is therefore not an identifier within different versions of the metafile. (A constant unique identifier for the observations of the metafile is the combination of the metafile variables *datenfile* and *varname*).
stcohor	Start cohort	stringVar Acronym of the start cohort (SUF SC#). Constructed from the string "SUF" and the first 3 characters of the data file name.
datenfile	Source file	stringVar Name of the source file.
dsign	Data signature (date)	stringVar Shows whether the data has been changed since the last time a data signature was assigned (originally by LifBi). See Stata command: *help datasignature* .
nobs	No. of observations in source file	numVar, range: Integers > 0 Number of observations in the source file.
nvrs	No. of vars in source file	numVar, range: Integers > 0
nvrsnm	No. of vars with non-miss values in source file	numVar, range: Integers >= 0
varname	Variable name	stringVar Name of the source variable.
varlab_de	variable label DE	stringVar German variable label of the source variable. Note: NEPS-Metafile.do determines the language versions of the source file and generates a variable for each of them containing the source variable label in the respective language.
varlab_en	variable label EN	stringVar English variable label of the source variable.

Variable	Variable label	Description
ivarname	Varname in questionnaire	stringVar Variable name in the questionnaire, taken from the "characteristics" of the source file. Empty if no variable name from the questionnaire is specified in the characteristics.
iquestno	Question no. in questionnaire	stringVar Question number in the questionnaire, taken from the "characteristics" of the source file. Empty if no question number is specified in the characteristics.
questde	German question text	stringVar German question text in the questionnaire, taken from the "characteristics" of the source file. Empty if no German question text is specified in the characteristics.
questen	English question text	stringVar English question text in the questionnaire, taken from the "characteristics" of the source file. Empty if no English question text is specified in the characteristics.
ofilter	Output filter	stringVar Output-filter-instruction, taken from the "characteristics" of the source file. Empty if the output-filter instruction is not specified in the characteristics.
afill	Autofill instruction	stringVar Autofill instruction, taken from the "characteristics" of the source file. Empty if the autofill instruction is not specified in the characteristics.
type	storage type	stringVar Storage type of the source variable.
isnumeric	whether numeric or string	numVar, range: 1, 0 Flag: source variable is numeric/not numeric.
nonmis	No. of non-miss obs in var	numVar, range: Integers >= 0 Number of observations of the source variable with values that are not: <nepsmissings> or <sysmis> (with numvars) Empty or <nepsmissings> (with stringVars) Note: In interval-scaled variables of the SUF, the values declared as <nepsmissings> may occur as valid values. For this reason, the metafile indicators basing on the inclusion or exclusion of the <nepsmissings> may have a (minor) error for interval-scaled source variables.

Variable	Variable label	Description
sysmis	No. of system-miss obs in var	numVar, range: Integers >= 0 Number of system-missing observations (., .a, .b….z). With string vars: -200
uniqvals	No. of levels in var	numVar, range: Integers > 0 Number of unique values (including missing codes and <sysmis>. With string vars: Number of diverse entries (including <empty>)
uniqvalpos	No. of levels in var >= 0	numVar, range: Integers >= 0 Number of unique values without <sysmis> and missing codes. With string vars: Number of diverse entries which are neither empty nor hold a NEPS missing code. See above, note to nonmis.
valuniq	Value if only 1 level	numVar, range: -900, -200, values of the source variables Value of the source variable if *uniqvals* == 1 -900 if uniqvals > 1 -200 with stringVars
valuniqpos	Value if only 1 level >= 0	numVar, range: -900, -200, positive values of the source variables Value of the source variable if *uniqvalpos* == 1 -900 if uniqvalpos != 1 -200 with stringVars
nvalzero	Frequency of value 0	numVar, range: Integers >= 0 With stringVars: frequency of empty strings
valmin	Minimum value of var	numVar, range: -200, values of the source variables Decimals rounded to 2 decimal places -200 with stringVars Note: *valmin* is <sysmis> if the source variable is <sysmis> allover.
valminp	Minimum value of var >= 0	numVar, range: -900, -200, values of the source variables >= 0 Minimal value of the source variable that is >=0 (= minimal non-missing value). -900 *valmax* < 0 (no non-missing value in the source variable) -200 with stringVars See above, note to *nonmis*.
valmax	Maximum value of var	numVar, range: -200, values of the source variables Decimals rounded to 2 decimal places Without <sysmis>, unless <sysmis> is the only value of the source variable (in that case, *valmin* is also <sysmis>). -200 with stringVars

Variable	Variable label	Description
vlabmax	Maximum labeled value	numVar, range: -200, values of the source variables -200 if no value label set is assigned NOTE: the indicator relates to the defined labels, not to the actual values.
vallabset	Name of value label set	stringVar Name of the attached value label set. "-200" if no value label set is attached
undoc	No. of unlabeled obs (labeled vars only)	numVar, range: -9, -2, Integers >= 0 Number of observations of the source variable in non-labeled categories, (when a value label set is assigned to the variable). -900 the source variable has a value label set, but only missing codes and possibly value 0 are labeled. -200 the source variable has no attached value label set.
strmin	Min. length of string var	numVar, range: -2, Integers >= 0 Minimal length of non-empty entries (= 0 if all entries in the source variable are empty). -200 with numVars.
strmax	Max. length of string var	numVar, range: -2, Integers >= 0 Maximal length of non-empty entries (= 0 if all entries in the source variable are empty). -200 with numVars.
wind	Wave indicator	stringVar Reports the wave indicator in the source used to generate the metafile-variables *nwaves*, *wavemin*, *wavemax*, and *wav1* - *wav#* (with NEPS data currently always: *wave)* -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*)
nwaves	No. of waves	numVar, range: -1, Integers >= 0 Number of waves in which the source variable is surveyed. Generated on the basis of the priority list of possible wave indicators defined in the parameter definition section of *NEPS-Metafile.do* 0 the source file contains a wave indicator, but the source variable has no non-missing values -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*)

Variable	Variable label	Description
wavemin	Earliest wave	numVar, range: -9, -1, Integers >= 0 & <= of the actual data release Earliest year in which the source variable was surveyed (i.e. the minimum value of the relevant wave indicator). -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*) -9 *nwaves* == 0 (i.e. the source file contains a wave indicator, but the source variable has no non-missing values)
wavemax	Latest wave	numVar, range: -9, -1, Integers >= 0 & <= of the actual data release Latest year in which the source variable was surveyed (i.e. the maximum value of the relevant wave indicator). -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*) -9 *nwaves* == 0 (i.e. the source file contains a wave indicator, but the source variable has no non-missing values
waves	Waves where var has nonmiss values	stringVar Contains, as a string, the combination of all waves in which the source variable has non-missing values. Each of the waves is represented with its specific code. -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*) 0 the source variable was not surveyed or has no non-missing values in any of the waves
wav1 - wav#	--	numVars, range: -1, 0, 1 Flag variables for each wave. 1 the source variable has nonmissing values in the respective wave 0 the source variable was not surveyed or has no non-missing values in the respective wave -1 the source file contains none of the wave indicators of the priority list (macro *syrlist*)

Variable	**Variable label**	**Description**
val1 - val100	--	numVars, range: -100, -200, -900, <sysmis>, values of the source variable val1-val100 show the actually observed levels of the source variables (only numVars with a maximum of 100 non-missing levels) -100 numVar has more than100 non-missing levels -900 numVar has not more than 100 levels, but all are missing codes -200 stringVars (The values of stringVars are recorded in the metafile-variables *lab1-lab100*, because *val1-val100* are numVars and cannot hold strings) <sysmis> *val#*-variables that are not taken
lab1 - lab100	--	strVars lab1-lab100 show the labels of the actually observed levels of the source variables (only variables with a maximum of 100 non-missing levels). "###" level without label, although the source variable has value labels (= undocumented levels, or scales where only the endpoints are labeled) "-100" source variable has more than100 non-missing levels "-200" source variable without attached value label set "-900" source variable has value label set and a maximum of 100 levels, but all values are missing codes With stringVars the metafile-variables lab# hold the unique values (levels) of the source variable. With more than 100 levels, the value of lab# is -100

5. Syntax examples on how to use the metafile

This section describes, among other things, how Stata string functions are used for data selection (data retrieval). From Stata 14, the names of string functions have a prefixed "u" (standing for "unicode"). For instance, the function *strpos()* becomes *ustrpos()* as of Stata 14. Therefore, if you are using a Stata version older than Stata 14, the syntax examples must be adapted accordingly.

Angle brackets in the syntax examples indicate placeholders. In the following syntax

```
edit <varlist> if <condition>
```

`<varlist>` is to be replaced by a list of variables in the file, and `<condition>` by a stated condition.

Working with the metafile consists mainly of retrieving data subsets (queries) from the overall metafile, viewing these subsets, and subjecting them to further procedures.

The data subset can be displayed onscreen. The selected subset is also available in the working memory for further processing, using Stata commands.

Both procedures are explained in the next section, which includes examples of use.

5.1 How to generate simple views or queries

Using the Stata data editor, the metafile can be viewed as a spreadsheet. Given the size of the metafile, it is much more covenient for users to restrict the displayed information to the data that is currently relevant.

To display the working file onscreen, Stata offers "browse" and "edit" modes. In edit mode, users can manually change values or copy them to the clipboard: in browse mode users can only view the data.

It is advisable to store commands to generate a view in the syntax editor and save them as a do-file, in order to make the procedure reproducible.

5.1.1 Change the display format of variables in the data editor

While viewing the metafile in the data editor, it is often more convenient to decrease the width of the variables, so that more variables are visible on the screen without as much scrolling. Examples:

```
format datenfile %20s
format varname ivarname %14s
format iquestno %10s
format type %4s
format varlab_de varlab_en %60s
format questde questen %60s
format nvrs %7.0g
```

➔ changing the display format of variables does not affect the contents of variables. The format of each variable is shown in the properties window.

5.1.2 Select variables and observations

```
edit <varliste> if <condition>
```
or:
```
browse <varliste> if <condition>
```

➔ The use of *edit* instead of *browse* enables data to be copied to the clipboard.

➔ With long variable lists it is more convenient to "click" the variable names to the command window and copy them via the clipboard to the syntax editor, rather than typing them.

Example 1: Generate corresponding lists of the SUF variable names and the variable names in the questionnaire

```
ed lfn datenfile varname ivarname iquestno varlab_en
```

The resulting contents of the data editor can be copied via the clipboard to an empty Excel spreadsheet or to a Word file. In Word the copied contents of the data editor is inserted as tab delimited text, which can be easily converted to a Word table.

You can also save the actual contents of the data editor as a Stata file:

```
preserve
keep lfn datenfile varname ivarname iquestno varlab_en
save "<Pfad>/varname_ivarname_SC6", replace
restore
```

➔ with preserve / restore you can save a subset of the data and after that continue to work with the complete metafile.

Example 2: Generate a table of the data files and their number of variables and observations

```
sort datenfile varname
capture drop h1
by datenfile: gen byte h1 = cond(_n == 1, 1, 0)
ed datenfile nvrs nobs if h1 == 1
```

This example only requires indicators which are constant within a source file. Therefore the first observation of each source file is marked with the auxiliary variable h1, which is then used as a selection criterion. The syntax component *cond(...)* means: "assign to h1 the value 1 if _n==1 (i.e. the first observation), otherwise value 0". Because of the instruction *by datenfile* _n has the value 1 in the first observation of each data file and counts up to the last observation of the file.

After running the above syntax the resulting view can be transmitted, as described above, to Excel, Word, a text editor, or saved as a Stata file:

```
preserve
keep if h1==1
keep datenfile nvrs nobs
save ""<Pfad>/datenfiles_nobs", replace
restore
capture drop h1
sort lfn
```

➔ The record number *lfn* in the metafile is used to restore the original order

Example 3: Generate a flag variable for source variables whose levels include level -4

The following example uses the metafile variables *val1 - val100.* Therefore it can be applied to numeric source variables with <= 100 levels, and to values which are not declared as NEPS missings (see chapter 4.2 for the documentation of the metafile variables *val1 - val100* and chapter 3.2 for the declaration of the NEPS missings).

```
capture drop t1
gen byte t1 = 0
forvalues n = 1(1)100  {
    replace t1 = 1 if val`n' == -4
}
fre t1
list datenfile varname if t1==1
```

5.1.3 Use string functions to select data subsets

Since the metafile contains many string variables, the fundamental string functions for data selection are presented below by examples.

Example 4: Selection of cases based on a specific string within a string variable

```
…if ustrpos(varlab_de, "erwerbs") > 0
```

The return value of the function ustrpos() is the position at which the specified string ("erwerbs") is found first within the specified variable (*varlab_de*). For instance, the condition selects cases where varlab_de is "Nebenerwerbstätigkeit" or "nicht erwerbstätig".

Example 5: Selection of cases based on a specific string within a string variable, case insensitive

```
…if ustrpos(ustrlower(varlab_de), "erwerbs") > 0
```

In this example, the functions ustrpos() und ustrlower() are nested. The expression says: ... if the string "(...)" is found within the contents of variable *(...),* which is converted into lower cases. The condition selects cases in which *varlab_de* is "Nebenerwerbstätigkeit" or "nicht erwerbstätig", or "Erwerbstätigkeit", for instance.

Example 6: Selection of cases where the contents of a string variable begins with a specific string

```
…if ustrpos(varname, "ts") == 1
```

or:

```
…if usubstr(varname, 1,2) == "ts"
```

The first line asks if the string "ts" is to be found starting from the first character of *varname.* The second line asks if the first two characters of *varname* are "ts".

Any leading or trailing spaces in the variables can be skipped with the usttrim() function:

```
…if ustrpos(ustrtrim(varname), "ts") == 1
…if ustrpos(ustrtrim(ustrlower(varlab_de)), "erwerbs") > 0
```

Example 7: Selection of cases where the contents of a string variable begins and ends with specific strings

```
…if usubstr(varname,1,2)=="ts" & usubstr(varname, ustrlen(varname)-1, 2)== "g1"
```

The condition checks, as in the previous example, the first two characters of *varname*. Since the contents of *varname* is of variable length, the last two characters cannot be determined by a constant position specification. Instead, the length of *varname* is used to determine the position of the last two characters. The return value of the function ustrlen(...) is the position of the last character of *varname*, and the next to last character has the position: length of *varname* minus 1.

➔ complex conditions are easier to handle and test when the components are stored in macros:

```
local cond1 `"usubstr(varname, 1, 2)=="ts"'
local cond2 `"usubstr(varname, ustrlen(varname)-1, 2)== "g1"'
ed if `cond1' & `cond2'
```

PLEASE NOTE: Strings that contain quotes must be enclosed by so-called compound quotes when assigned to a macro (see the yellow marked outer characters in the first two lines of the syntax above). Compound quotes are not required to expand these macros, as can be seen from line 3 of the syntax.

➔ For further details and more string functions see Stata command *help string functions.*

5.2 Complex selection methods for source variables using the metafile

Please note: the following examples are not significant in terms of substance; they serve only as a means to demonstrate complex selection methods. The examples build on each other, i.e,. a data subset is varied and used consecutively for the examples. These have been prepared with a metafile over all start cohorts, generated with the NEPS-SUF delivered in May/June, 2023.

Example 8: Select source variables covering a certain topic using the question number in the questionnaire.

```
ed if ustrpos(iquestno, "261") == 1
```

The above syntax selects all variables which have a question number in the questionnaire beginning with the characters "261".

Please note: the condition only captures variables that have a question number in the questionnaire. If the aim is to select *all* variables covering a topic, check in a second step the whole possible variable field by means of the record number *lfn* (see section 5.4.2).

Example 9: The subset from example 8 is to be reduced in a way that it only contains source variables surveyed in at least 5 waves since wave 3, and which have at least 500 non-missing cases.

NOTE—potential pitfall: You might be tempted to put the condition as follows:

```
… if wavemin >= 3 & nwaves >= 5
```

However, this would be wrong, as the condition excludes those variables that comply with the selection criterion but were also surveyed in waves earlier than the 3rd.

Instead, an auxiliary variable is generated that indicates how often a variable has been surveyed since the 3rd wave:

```
capture drop h1
egen h1 = anycount(wav3-wav11), values(1)
```

Thereafter, the auxiliary variable *h1* can be used to state the selection criterion:

```
ed if ustrpos(iquestno, "261") == 1 & h1 > 4 & nonmis >= 500
```

For the following, the conditions that produce the above subset are stored in a macro:

```
local cond1 `" ustrpos(iquestno, "261") == 1 & h1 > 4 & nonmis >= 500"'
ed lfn-iquestno varlab_en nobs nonmis uniqvalpos valuniqpos vallabset ///
nwaves-waves if `cond1'
```

➔ local macros have to be defined anew with each run of the syntax. You can define a global macro instead that stays alive throughout the whole Stata session:

```
global cond1 `" ustrpos(iquestno, "261") == 1 & h1 > 4 & nonmis >= 500"'
ed lfn-iquestno varlab_en nobs nonmis uniqvalpos valuniqpos vallabset ///
nwaves-waves if $cond1
```

Example 10: Show potentially similar variables in different start cohorts.

The subset from example 9 stays in memory, but is sorted differently. Furthermore, the order of the displayed variables is changed:

```
sort varlab_en stcohor datenfile
ed varname-iquestno varlab_en stcohor vallabset datenfile nobs nonmis ///
uniqvalpos valuniqpos nwaves-waves if $cond1
```

The inspection of the displayed data shows that same or similar variables which have a question number in the questionnaire beginning with "261" occur in several start cohorts. Indicators for sameness or similarity are the variable label, the question number in the questionnaire, the attached value label set, the number of non-missing levels, and the single levels (*varlab_en, iquestno, vallabset, uniqvalpos, val1 - val100*).

Example 11: Further reduction of the subset to the source variables which occur in each of the start cohorts 4, 5 and 6.

Note: The example presumes that variables with the same contents in the various start cohorts have the same question number in the questionnaire. This is usually true, but there are exceptions, for instance when particular topics have been shifted to different modules.

Step 1: Checking which start cohorts are present in the subset:

```
fre stcohor if $cond1
```

Step 2: Identification of the source variables in the subset which occur in each of the three start cohorts 4, 5 and 6:

```
global cond2 ///
`"( ustrpos(stcohor,"4")>0 | ustrpos(stcohor, "5")>0 | ustrpos(stcohor, "6")>0) "'
capture drop c1
gen byte c1 = cond($cond1 & $cond2, 1, 0)
sort c1 iquestno varlab_en stcohor
```

```
capture drop c2
by c1 iquestno varlab_en: ///
gen byte c2 = cond(stcohor != stcohor[_n-1] & c1 == 1, 1, 0)
capture drop stc
by c1 iquestno varlab_en: egen stc = total(c2)
```

Explanation: The conditions *cond1* and *cond2* are mirrored in the auxiliary variable *c1* (*c1* is 1 if *cond1* and *cond2* is true, else *c1* is 0). After that, the metafile is sorted by *c1*, question number, variable label, and start cohort. As *stcohor* is the last sorting criterion, the different starting cohorts of each group formed by *c1, iquestno, varlab_en* follow one another. The auxiliary variable *c2* is 1 whenever the start cohort changes within the group, else *c2* is 0. *stc* counts the number of changes within a group. Source variables occuring in each of the three start cohorts 4, 5 and 6 have the value 3 in *stc,* which can be verified by

```
edit stcohor datenfile varname varlab_de iquestno if stc == 3
```

5.3 Exporting elements of the data subset to the Stata output or to a text file

The easiest way to export the content of the data editor is to use the clipboard, as described above. A more sophisticated method to export subsets of the metafile to the Stata logfile or to a text file is to use Stata syntax.

Continuation of Example 11

In a first step we mark the subset using a flag variable and chose an appropriate sort order for the output. The use of a flag variable makes it easier to handle the subset.

```
capture drop f1
gen byte f1 = cond(stc == 3, 1, 0)
sort datenfile iquestno varname
```

Note: You could use `stc == 3` instead of `f1 == 1` in the following syntax, but at this point the use of a flag variable to identify a subset of the metafile is to be demonstrated.

After that, the selected characteristics are printed out as follows:

5.3.1 Printing to the Stata logfile

```
capture log close
log using "<path>/test1.log", replace
local n = _N
forvalues i = 1(1)`n'      {
    if f1[`i'] == 1    {
        local file = datenfile[`i']
        local svn = ivarname[`i']
        local vn = varname[`i']
        local qu = questen[`i']
        display "    "
        display "`file'  instr: `svn'   suf: `vn'"
        display `"`qu'"'
        display "   "
    }
}
capture log close
```

Explanation: In a loop over all observations of the metafile the program checks if the respective observation is part of the subset. If it is, the name of the source file, the variable name in the questionnaire, the variable name in the SUF, and the German question text are written to the log file.

➔ If the string that is to be printed to the logfile using the Stata command *display* contains quotes, the complete string has to be enclosed by compound quotes. for example:

```
local cond1 `" ustrpos(iquestno, "26") == 1 & h1 > 4 & nonmis >= 500"'
display `"cond1"'
```

5.3.2 Printing to an external text file

Printing to an external text file allows for a more sophisticated layout of the output:

```
capture file close d1

file open d1 using "<path>/test2.txt", write replace
file write d1 _n
file write d1 "#################################################################" _n
file write d1 "###  List of variables which meet criterion xyz and are    ###" _n
file write d1 "###  surveyed in at least 5 waves since the 3rd wave       ###" _n
file write d1 "#################################################################" _n
file write d1 _n

local n = _N
forvalues i = 1(1)`n'           {
    if f1[`i'] == 1         {
        local file = datenfile[`i']
        local svn = ivarname[`i']
        local vn = varname[`i']
        local qu = questen[`i']
        file write d1 " `file' " _n
        file write d1 "     Variable name: `vn' " _n
        file write d1 "     Varname in the questionnaire: `svn' " _n
        file write d1 "     `qu' " _n
        file write d1 _n
    }
 }
file close d1
```

Explanation: The external text file is defined by the *file open* command, followed by a text to indicate the content of the text file. Then, in a loop over all observations of the metafile, the selected information is written to local macros and transferred to the external text file. You can shape the output using blanks, and *_n* to create empty lines.

➔ If the string that is to be printed to the external text file using the Stata command *file write* contains quotes, the complete string has to be enclosed by compound quotes, for example

```
file write d1 `"  replace h1=0 if  ustrpos(stcohor, "1") > 0  "' _n
```

Enclosing the text in compound-quotes when it is not required, does not cause an error, nor any other problem.

Excerpts of the resulting text file are shown in Figure 2.

Figure 2: View of the text file generated with the above Stata syntax

```
############################################################
###  List of variables which meet criterion xyz and are    ###
###  surveyed in at least 5 waves since the 3rd wave       ###
############################################################


 SC4_spEmp_D_13-0-0
     Variable name: ts23550
     Varname in the questionnaire: etmod
     [AUTO] Episode mode

 SC4_spEmp_D_13-0-0
     Variable name: ts23204
     Varname in the questionnaire: etdbs
     What professional status did or do you have there exactly?

 SC4_spEmp_D_13-0-0
     Variable name: ts23209
     Varname in the questionnaire: etselb1
     Were you self-employed in a freelance profession, e.g., physician, lawyer or architect or self-employed in agriculture or another area?

 SC4_spEmp_D_13-0-0
     Variable name: ts23211
     Varname in the questionnaire: etselb3
     What kind of self-employment was that in the beginning?

 SC4_spEmp_D_13-0-0
     Variable name: ts23214
     Varname in the questionnaire: etaus
     What kind of employment is/was it?

 SC4_spEmp_D_13-0-0
     Variable name: ts23215
     Varname in the questionnaire: etba
     Was this a one-euro job or an occasional job?

 SC4_spEmp_D_13-0-0
     Variable name: ts23217
     Varname in the questionnaire: etsais
     Did you work there as a seasonal worker?
```

Stata do-files can be generated in the same way, see chapter 5.5.

5.4 More tips on data selection

5.4.1 Using a flag variable to mark selected observations

In the previous example a flag variable is used to mark whether an observation meets the conditions *cond1* and *cond2*. There are more cases where the use of a flag variable makes sense. Some selection sets are difficult to summarize in a clear selection criterion. For example, a selection criterion may select some variables that do not belong to the desired hit list. Or the selection criterion is already very complex, but further conditions have to be added to achieve the desired result.. Here it is useful to generate a flag variable that indicates whether a variable belongs to the selection set.

While it is possible to change the flag variable manually in the data editor, it is strongly recommended not to do so. Instead, you should document all steps in a syntax file (Stata command *replace*), in order to make the data processing reprocucable and traceable in the future.

If, however, the only purpose of the flag variable is to write out a do-file in which the flagged variables are listed, the resulting do-file is also the process documentation. In this case the flags can be set manually in the data editor, provided that you do not need to repeat the process.

5.4.2 Using the consecutive observation number (*lfn*) to identify observations that belong to the same topic

When aiming to select variables within a source file that belong to a particular subject area, it is recommended to check if the record number of the subset shows gaps. These may indicate that source variables have been inadvertently skipped. This can happen if a selection criterion is used which is not available for all source variables (e.g. *iquestno*, the question number in the questionnaire).

To control for this, display the related observations in the data editor, that is, the observations with record numbers that lie within the margins of the record numbers of the selected subset, plus some adjacent observations. If the lowest record number of your subset is 10889 and the highest 10960, display all observations with a record number from 10800 to 11000 in the data editor, and check if relevant source variables are hidden in the gaps of the record numbers in the subset. If this is the case, add them to your data subset by setting a flag, as described in the preceding section.

5.5 Collecting the variable names in the selected subset for further use

The names of the source variables that have been selected from the metafile can be collected and used in Stata commands or in Stata do-files. Users may for example wish to use only some variables of a source file, rename a lot of source variables, or compose a data file for analyses from variables of various source files.

To this end we apply the Stata-command *levelsof*, which stores the distinct values of a variable in a macro list. After all, the distinct values of the metafile variable *varname* are just the names of the NEPS source variables.

➔ ATTENTION: if a variable has a great number of levels (as is the case with *varname*) the maximum length of macros in your Stata application might be exceeded. See Stata command *help limits* on how to determine and increase the maximum macro length.

5.5.1 Case 1: All required variables are in the same source file

In a first step we assume that the source variables in the selected subset are to be found in one and the same source file. The subset of the metafile has been marked by the flag variable *f2* previously:

```
quietly levelsof varname if f2 == 1, clean local(vars)
global myvars "`vars'"
display "${myvars}"
```

Explanation: The first line of the syntax assigns the names of the flagged source variables to the local macro *vars*. The option *clean* achieves the elimination of the quotes that otherwise enclose the variable names in the macro. However, *levelsof* can only produce local macros. To make the list of the source variable names available outside of this procedure, it is assigned to the global macro *myvars* in the second line of the syntax.

After that, the names of the selected variables are available for further processing.

The list can be supplemented by additional variables, such as weight factors or identifiers. These can be marked using a flag variable, or added to the macro list directly:

```
global allmyvars "ID_t splink sample ${myvars}"
display "${allmyvars}"
```

The resulting macro list can now be used as a variable list when working with the related source data. It can, for example, load a subset of variables from the data file <name of the data file> into the working memory:

```
use ${allmyvars} using <name of the data file>, clear
```

5.5.2 Case 2: The required variables are in several different source files

The following example demonstrates how to proceed when the selected variables are located in several different source files.

The aim is to compose a particular file for analyses: Selected information from different NEPS episode files and the Basic file are to be merged to the Biography file. To this end, a Stata do-file is generated by Stata syntax from the metafile information.

The example uses only data from the start cohort 6.

```
use <metafile>, clear
keep if ustrpos(stcohor, "SC6") > 0
```

In a first step, the desired variables are selected using the metafile and a flag variable.

So that the example is reproducible by the reader, the flag variable is set with a series of *replace* commands. In practise this is set manually. It is recommended that you create a convenient view of the required metafile information using *edit* and *format*-commands, for example

```
format datenfile %30s
format varname %14s
format varlab_de varlab_en %50s
format waves %20s
format uniqvalpos %7.0g
format nonmis %10.0g

edit datenfile f1 varname varlab_de nobs nonmis uniqvalpos waves varlab_de ///
 val1-val100 lab1-lab100
```

Generating the flag variable:

```
capture drop f1
gen byte f1 = 0
```

Setting the flag-variable *f1* by *replace* commands:

```
replace f1 = 1 if varname == "t405000_g2" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t510011_g1" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t700001" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t70000y" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t731301_g1" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t731351_g1" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "t741001" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx20001" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx20002" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx20003" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx27000" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx28101" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "tx28102" & datenfile == "SC6_Basics_D_13-0-0"
replace f1 = 1 if varname == "ts23201_g2" & datenfile == "SC6_spEmp_D_13-0-0"
```

```
replace f1 = 1 if varname == "ts23204_ha" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23219_g1" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23223_g1" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23411_v1g1" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23241" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23310" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "ts23320" & datenfile == "SC6_spEmp_D_13-0-0"
replace f1 = 1 if varname == "th27100" & datenfile == "SC6_spParLeave_D_13-0-0"
replace f1 = 1 if varname == "th27101" & datenfile == "SC6_spParLeave_D_13-0-0"
replace f1 = 1 if varname == "ts11103_g1" & datenfile == "SC6_spSchool_D_13-0-0"
replace f1 = 1 if varname == "ts11204_ha" & datenfile == "SC6_spSchool_D_13-0-0"
replace f1 = 1 if varname == "ts11209" & datenfile == "SC6_spSchool_D_13-0-0"
replace f1 = 1 if varname == "ts25205" & datenfile == "SC6_spUnemp_D_13-0-0"
replace f1 = 1 if varname == "ts25206" & datenfile == "SC6_spUnemp_D_13-0-0"
replace f1 = 1 if varname == "ts25207" & datenfile == "SC6_spUnemp_D_13-0-0"
replace f1 = 1 if varname == "ts25209" & datenfile == "SC6_spUnemp_D_13-0-0"
```

Thereafter the do-file to join the selected variables is generated. The syntax determines the involved source files, then loops over these while compiling and out-writing the diverse elements of the do-file.

```
capture file close d1
file open d1 using "<path/name of the do-file to be created>", write replace

local path `"<path to the NEPS data files>"'
local q = char(34) /* double quotes */
local str1 "use `q'`path'SC6_Biography_D_13-0-0.dta`q', clear" /* Masterfile */
file write d1 `"`str1'"' _n
file write d1 _n

/* Determine source files and index of the loop */
quietly levelsof datenfile if f1 == 1, clean local(files)
local nf : word count `files'

/* compile the merge-commands in a loop over the source files */
forvalues i = 1(1)`nf'    {
    local f : word `i' of `files'
    quietly levelsof varname if f1==1 & datenfile == "`f'", clean local(vars)
    local keyvars "ID_t splink"
    local mtype "1:m"
    local keepus "keepusing(subspell `vars')"
    local keepstr "keep if _merge == 1 | (_merge == 3 & subspell == 0)"
    local dropstr "drop _merge subspell"
    if ustrpos("`f'", "Basics") > 0 {
        local keyvars "ID_t"
        local mtype "m:1"
        local keepus "keepusing(`vars')"
        local keepstr "keep if _merge == 3"
        local dropstr "drop _merge"
    }
    file write d1 `"merge `mtype' `keyvars' using `q'`path'`f'`q', `keepus'"' _n
    file write d1 `"`keepstr'"' _n
    file write d1 `"`dropstr'"' _n
    file write d1 _n

}
file write d1 `"save `q'`path'myworkfile`q', replace "'

/* ending with a blank line (mandatory) */
file write d1 _n
file close d1
```

Explanation:

The Biography-file is the masterfile to which the selected data of the Basic file and the diverse episode files are merged.

Initially, the do-file is defined, which is a text file that is addressed by Stata with a file handle, here called *d1* (see section 5.3.2). The do-file can be named *mergetogether.do*, for example.

The path to the NEPS source files is assigned to a local macro because it is needed several times throughout the syntax. In the example, the resulting analyses file is put out to the NEPS source file directory. If you want it to be put out to a different directory, you can either define a second path macro or adapt the *save* command at the end of the syntax accordingly.

The command to use the master file is written to the first line of the created do-file.

Next, the required *merge* commands are compiled in a loop over the involved data files.

The data files are determined using the command *levelsof*, (see section 5.5.1). The index of the loop is obtained by counting the number of data files in the macro list *files.*

Within the loop, the file-specific variable lists are assigned to the macro *vars*, and the other components of the *merge* commands to the macros *keyvars, mtype, keepus, keepstr* and *dropstr*.

The Basic data file must be treated separately using an *if* construction, since the key variables and other components of the *merge* commands are different from the commands for the episode files.

In the last part of the syntax within the loop, the defined do-file is filled in line by line.

The *save* command for the resulting analyses file and a closing blank line (which seems to be mandatory to make the do-file executable) are written to the do-file outside the loop.

➔ It is often difficult to put the syntax for double quotes within macro contents correctly. A workaround is to assign the double quotes (ASCII Code and Unicode 34) to a macro, as is done in the above syntax.

➔ Pay attention to the enclosing compound quotes in the *file write*-commands, they are required!

➔ The last line that is written to *d1* before it is closed has to be a blank line - *file write _n* - to make the do-file executable.

➔ If the lines in the resulting do-file are too long to fit the screen width, you can include line breaks manually (add the string /// to the end of the line that is to be continued). In principle this could be done by syntax, but it is hardly worth the effort of programming. (In the actual Stata version 17 line breaks seem not to be required any more).

The examples of Stata syntax in this chapter demonstrate the use of the metafile generated by *NEPS-Metafile.do.* They are intended to prompt further applications of your own.

A last tip:

➔ If you adapt the examples to your own requirements, it can be helpful to enclose your syntax in the *trace on* / *trace off* commands:

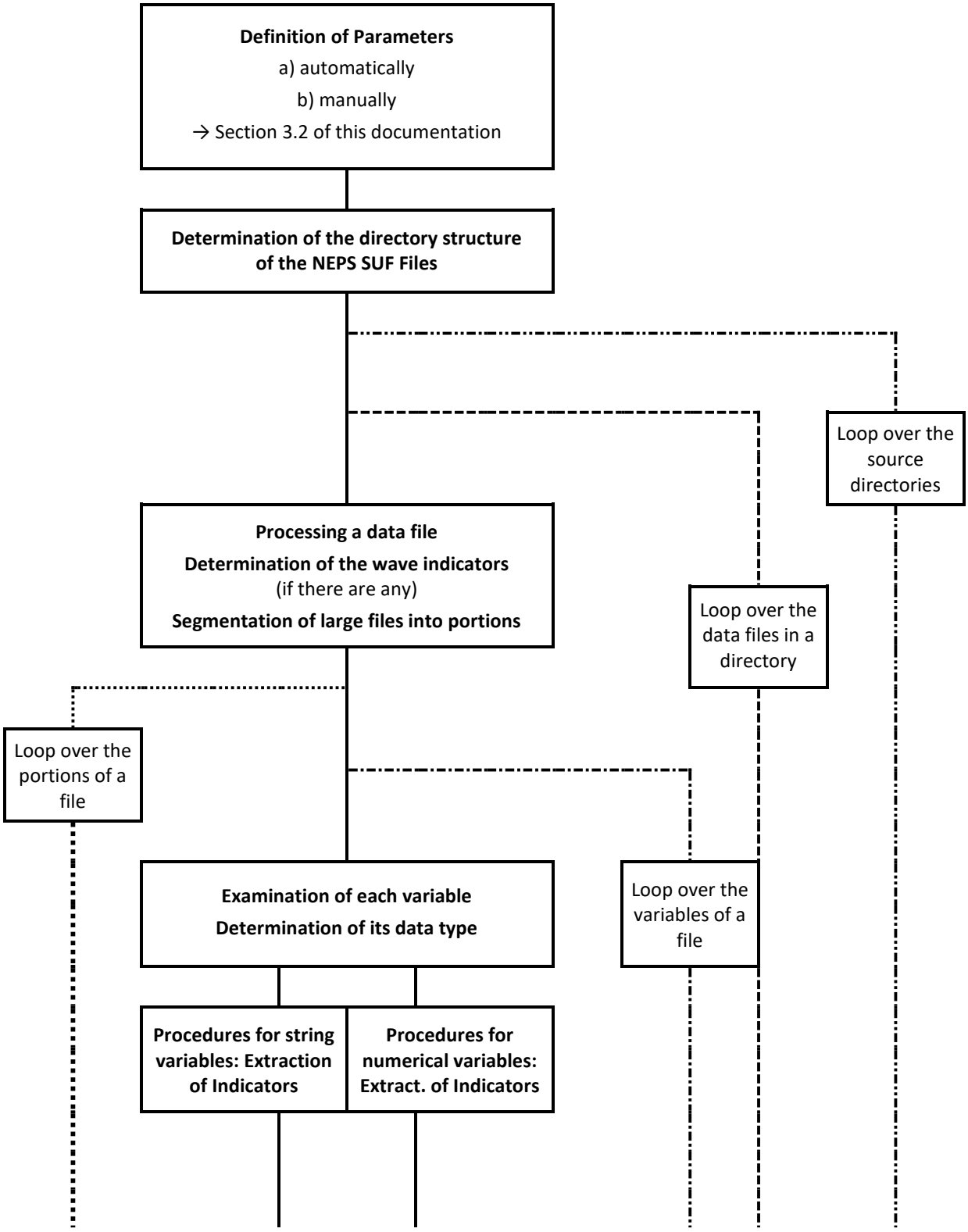
```
set trace on
set tracedepth 1

<Syntax>

set trace off
```

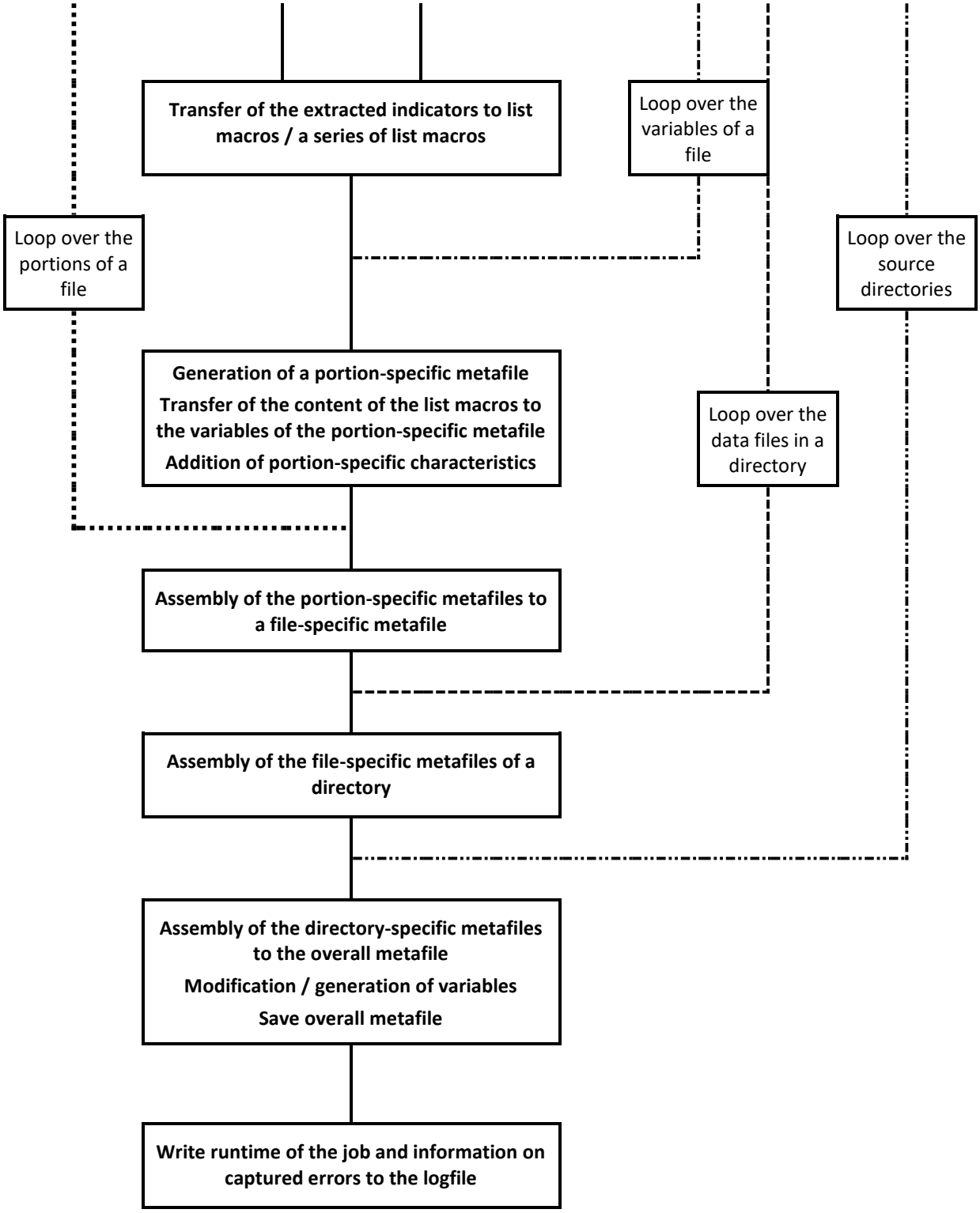
By doing so, locations of possible errors are marked in the output.

6. Appendix 1: Flow chart of NEPS-Metafile.do



(Continued on next page)

(Continued from previous page)



7. Appendix 2: Complete syntax of NEPS-Metafile.do

```
/*  NEPS-Metafile-Do-File to generate a Metafile for the NEPS-Scientific-Use-Files
    Version 03-00
    Author: Klaudia Erhardt
    last updated: 2023-09-27
    feedback and questions to: erhardtk@gmx.de*/
    NEPS-Metafile.do is licensed under CC BY-NC-SA 4.0 (http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en)

set output error

/*
    Version history

    V02 Characteristic NEPS_instname replaced by characteristic NEPS_alias
        IF YOU ARE WORKING WITH EARLIER SUF DATA THAT STILL HAS THE CHARACTERISTIC "NEPS_instname", search
        and replace the string [NEPS_alias] with [NEPS_instname]

    V03 Added new variable dsign to the metafile, containing the data-signature of the source file

    Please note:

    Changes to adapt the do-file to your local environment and preferences are to be done in part B)
    Changes elsewhere in the do-file neither allowed nor necessary !!!

Approximative runtime of this job:
ca. 0:40 - 2:30 hrs when applied to all SUFS of the 4 start cohorts
(depends on endowment of the computer and workload of the server)
*/

/*##############################################################################################################
##############################                                          ##############################################
##############################  A) AUTOMATICALLY DEFINED PARAMETERS -   ##############################################
##############################      no changes please!!!                ##############################################*/

clear
set more off
set varabbrev off, permanently
```

```
/* Measuring the runtime of the program */
timer clear
timer on 1
tempname time
scalar `time' = 0

/* prefix for stata14+ string functions */
local u ""
if c(stata_version)>=14 {
    local u "u"
}

/* Date specification, is appended to resulting files' names */
local datum : display %td_CY-N-D  date("$S_DATE", "DMY")
local datum = `u'strtrim("`datum'")

/*##########################################################################################################
##############################                                          ###################################################
##############################  B) MANUALLY DEFINED PARAMETERS          ###################################################
##############################      (users' input)                      ###################################################*/

/* locals in this section are defined with examples, to be replaced by the user.
See chapter 3.2 of 'NEPS-Metafile-Do - a Do-File to Generate a Metafile on the Scientific Use Files of the NEPS' */

/* ###########      1.) Source files                                    ############################  */

local files "*.dta"         /* selection criterion within the files of the source directories
                       you may use wildcards to include only a subset of files.
                       If empty or not a valid Stata file specification, default "*.dta" is put into effect. */

/* Source directories */

local rootdir "M:\user\myname\SUF\Daten\"     /* Main directory containing either the SUF-files or Subdirectories */
local subd "*"             /* Wildcard to denominate the SUF-Sub-Directories, if there are any, may stay empty if
                              source files are in rootdir */
local subsubd "Stata14"    /* Data directory within each SUF-directory - has to be named alike for all
                              SUF-directories, may stay empty if source files are in rootdir or in subd */
```

```
/* ############     2.) Resulting files                                  ############################  */

/* Paths to resulting files' directories - both are mandatory, but may contain the same path specification
   Must be different from source directory, otherwise in a next run the resulting metafile will be included */

local pfad1d "M:\user\myname\output"  /* local directory for resulting metafile */
local pfad1o "M:\user\myname\output" /* local directory for logfile */

/* Definition of resulting file names und data labels  */
local we "NEPS-SUF_" /* we and we1 are labels for version and start cohorts. Will be appended to the
                        resulting files' names. Must comply with the file name conventions of Stata,
                        i.e. no period characters. May stay empty */
local we1 "SC1-2-3-4-5-6_"

local result "metaf_`we'`we1'`datum'"                                          /* name of the resulting metafile */
local reslab "metafile NEPS-SUF from NEPS-metafile_v03-00.do"   /* label to be attached to the resulting metafile,
                                                                   may stay empty */
local lg "metaf_`we'`we1'`datum'.log"                           /* name of the resulting log-file */


/* ############     3.) Further specifications                           ############################  */

/* Specification of wave indicators    */
local wave1 "1"  /* earliest wave   */
local wave2 "18"  /* newest wave in any of the included source files, wave1 <= wave2 is mandatory  */

/* Priority list of possible wave indicators: highest priority first. Usually you don't have to change this
    but if ever additional wave indicators turn up in the data you may add them to the priority list */

local syrlist "wave"

/*  Specification of missing values                    */

local nepsmissundc "-51 -32 -19 -4"            /* missing codes not contained in nepsmiss.ado from LifBi-Nepstools */
local nepsmissdc "-99/-90 -56/-52 -29/-20 -9/-5" /* missing codes, contained in nepsmiss.ado from LifBi-Nepstools */

local nepsmissings "`nepsmissdc' `nepsmissundc'"  /* Don't change this */

/*  Size of portions (usually you don't have to change this)  */
```

```
local port 50   /* Number of variables per portion for segmented processing of the data files.
                   Larger portions result in a longer runtime of the job. */


/*############################                                          ##############################################
##############################  END: DEFINITION OF PARAMETERS           ##############################################
############################                                            ##############################################
#########################################################################################################################*/

/* no changes in the follwing syntax required !!!!!!!!!!!!!!!                                                          */

/*#########################################################################################################################
############################                                            ##############################################
############################  START OF THE PROGRAM                      ##############################################
############################                                            ##############################################*/

/* ##########    Default definition of macros  - no changes please !!! ######################### */

/* path standardization  */

local pflist "data rootdir subd subsubd pfad1d pfad1o"

local nps : word count `pflist'

forvalues n = 1(1)`nps'    {

    local mstr : word `n' of `pflist'
    local str "``mstr''"
    while `u'strpos("`str'", "\") > 0 {
        local str = `u'subinstr("`str'","\", "/", 1)
    }
    if `u'strpos("`str'", "//") > 0    {
        local tstr1 = `u'substr("`str'",1,2)
        local tstr2 = `u'substr("`str'",3,`u'strlen("`str'")-2)
        while `u'strpos("`tstr2'", "//") > 0  {
            local tstr2 = `u'subinstr("`tstr2'","//", "/", 1)
        }
        local str = "`tstr1'" + "`tstr2'"
    }
    local str = `u'strreverse("`str'")
```

```
    if `u'strpos("`str'", "/") == 1 {
        local str = `u'subinstr("`str'", "/", "", 1)
    }
    local str = `u'strreverse("`str'")
    local `mstr' "`str'"
}

/* set default values */

if `u'strlen("`files'") == 0 | `u'substr(`u'strreverse("`files'"), 1, 4) != "atd."        {
    local files "*.dta"
    local message "    " _n ///
        "the default file specification `files' is used, as there has been made   " _n ///
        "no (valid) file specification  " _n ///
        "    "
        display as err "`message'"
        local message ""
}

if strlen("`port'") == 0   {
    local port 50
}
if `port' == 0     {
    local port = 50
}

local nw = `wave2'-`wave1'+1 /* ATTENTION: don't change the position of this specification: it must come before the
                                following definition of wave1 and wave2! Otherwise a variable "wav0" will be
                                generated */
if `u'strlen("`wave1'") == 0   {
    local wave1 "0"
}
if `u'strlen("`wave2'") == 0   {
    local wave2 "0"
}
if `wave2' - `wave1' < 0   {
    local abbruch = 1
        local message "    " _n ///
        "Error in the wave specification: wave1 is greater than wave2          " _n ///
            `"The Do-file stops. Please correct the macros "wave1" and "wave2" in Section B)  "'   _n ///
```

```
            "of the parameter definition    "   _n ///
            "                    "
        display as err "`message'"
        exit
}
/* Initialization of nsyrl in case wave1 and wave2 are defined empty */
local nsyrl 0

/* ##########    Identification of the input and output subdirectories and files               ############## */

/*   a) Output Directories    */
if `u'strlen("`pfad1d'") > 0  {
    quietly capture cd `"`pfad1d'"'
    if _rc == 0{
        local pfad1d "`pfad1d'"
        local abbruch = 0
    }
    if _rc != 0{
        local abbruch = 1
        local message1 "    " _n ///
        "You named an output-directory for the resulting metafile that does not exist.  " _n ///
            `"The do-file stops. Please correct the macro "pfad1d" in Section B)"'   _n ///
            "of the parameter definition    "   _n ///
            "    "
    }
}
if `u'strlen("`pfad1d'") == 0 {
    local abbruch = 1
    local message2  "    " _n ///
    "You did not name an output-directory for the resulting metafile   " _n ///
        `"The do-file stops. Please correct the macro "pfad1d" in Section B)"'   _n ///
        "of the parameter definition.    "   _n ///
        "    "
}
if `u'strlen("`pfad1o'") > 0  {
    quietly capture cd `"`pfad1o'"'
    if _rc == 0{
        local pfad1o "`pfad1o'"
        local abbruch = 0
    }
```

```
    if _rc != 0{
        local abbruch = 1
        local message3 "    " _n ///
        "You named an output-directory for the logfile that does not exist.  " _n ///
            `""The do-file stops. Please correct the macro "pfad1o" in Section B)"'   _n ///
            "of the parameter definition.    "   _n ///
            "    "
    }
}
if `u'strlen("`pfad1o'") == 0 {
    local abbruch = 1
    local message4 "    " _n ///
    "You did not name an output-directory for the logile   " _n ///
        `""The do-file stops. Please correct the macro "pfad1o" in Section B)"'   _n ///
        "of the parameter definition.    "   _n ///
        "    "
}

if `u'strlen("`message3'") > 0{
    display as err "`message3'"
}
if `u'strlen("`message4'") > 0{
    display as err "`message4'"
}
if `u'strlen("`message1'") > 0{
    display as err "`message1'"
}
if `u'strlen("`message2'") > 0{
    display as err "`message2'"
}

if `abbruch' == 1  {
    local message1 ""
    local message2 ""
    local message3 ""
    local message4 ""
    exit
}
```

```
/*   b) Source Directories   */

if `u'strlen("`subd'") == 0  & `u'strlen("`subsubd'") == 0    {
    local data "`rootdir'"
    local switch = 0
}
if `u'strlen("`subd'") > 0 {
    local switch = 1
}
if `u'strlen("`subd'") == 0  & `u'strlen("`subsubd'") > 0     {
    local abbruch = 1
    local switch = -1
    local message6  "    " _n ///
           "For the source data files you specified a subdirectory of the SUF-directories, " _n ///
           "but you did not specify the SUF-directories. This is invalid.          " _n ///
           `""The do-file stops. Please correct the macro "subd" or the macro "subsubd" "'   _n ///
           " in Section B) of the parameter definition.  "    _n ///
                  "    "
}
if `switch' == 0   {
    if `u'strlen("`data'") > 0 {
        quietly capture cd `"`data'"'
        if _rc == 0 {
            local data "`data'"
            local k1 = 1
            local nrd = 1
            local infile : dir "`data'" files `"`files'"', respectcase
            local a : word count `infile'
            if `a' > 0  {
                local abbruch = 0
            }
            if `a' == 0 {
                local abbruch = 1
                local message "    " _n ///
            "In the source-directory `data' no Stata file has been found that corresponds          " _n ///
            "to your specification `files'              " _n ///
            `""The do-file stops. Please correct the macro "files" or the macro "rootdir" "'   _n ///
            " in Section B) of the parameter definition.  "    _n ///
                   "    "
            }
```

```
        }
        if _rc != 0{
            local abbruch = 1
            local message "    " _n ///
            "The source directory you specified has not been found.  " _n ///
            `""The do-file stops. Please correct the macro "rootdir" in Section B) "'   _n ///
            " of the parameter definition.  "    _n ///
                    "    "
        }
    }
    if `u'strlen("`data'") == 0   {
        local abbruch = 1
        local message "    " _n ///
        "You did not specify a source directory.  " _n ///
        `""The do-file stops. Please correct the macro  "rootdir" in Section B) "'   _n ///
        " of the parameter definition.  "    _n ///
        "    "
    }
}

if `switch' == 1   {

    /* Does rootdir exist? */
    if `u'strlen("`rootdir'") > 0 {
        quietly capture cd `"`rootdir'"'
        if _rc == 0{    /* rootdir has been found */
            local abbruch = 0

            /* If yes: do the SUF-directories in rootdir exist? */
            local alld : dir "`rootdir'" dirs "`subd'", respectcase   /* generat a list of the SUF-Directories */
            if `u'strlen(`"`alld'"') > 0  {
                local alld : list sort alld
                local nrd : word count `alld'
                local k1 = `nrd'
                local abbruch = 0
                local message "    " _n ///
                              `""The directories `alld' will be processed "'  _n ///
                              "    "
                noisily display "`message'"
                local message ""
```

```
/* If yes: Loop over the SUF-Directories:
   If subsubd is not specified, check if there are any Stata files corresponding to "files"
       in the SUF-Directories  and compile a list of SUF-Directories where there were found none.
   If subsubd is specified, check if it is unique, if it exists in each of the SUF-Directories,
       and check if there are any Stata files corresponding to "files".
       And compile lists of the Directories where subsubd is not unique or contains no Stata File
       defined by "files".
*/

if `u'strlen(`"`subsubd'"') == 0  {
    local nulliste ""
    forvalues l1 = 1(1)`k1'    {
        local dir1 : word `l1' of `alld'
        local infile : dir "`dir1'" files `"`files'"', respectcase
        local a : word count `infile'
        if `a' == 0 {
            local nulliste `"`nulliste' "`dir1'""'
        }
    }
    /* Program stops if there is not at least 1 SUF-Directory containing "files"-Stata-files */
    if `u'strlen(`"`nulliste'"') > 0  {
        local nnull : word count `nulliste'
        if `nnull' == `nrd'       {
            local abbruch = 1
            local message4  "    " _n ///
            `"In the Source-directories `nulliste'           "' _n ///
            "no Stata file corresponding to `files' has been found.            " _n ///
            `"The do-file stops. Please correct the macro  "subsubd" in Section B) "'   _n ///
            "of the parameter definition or check the structure of the  "    _n ///
            "Source directories."    _n ///
                        "    "
        }
        if `nnull' < `nrd'     {
            local messagex  "    " _n ///
            `"The source directories `nulliste' "'  _n ///
            "are excluded from being processed as they do not contain a Stata file   " _n ///
            "corresponding to `files' ."
            "    "
            noisily display "`messagex'"
```

```
        }
    }
}

if `u'strlen(`"`subsubd'"') > 0   {
    local nulliste ""
    local dupliste ""
    forvalues l1 = 1(1)`k1'    {
        local dir1 : word `l1' of `alld'
        local subdir "`rootdir'/`dir1'"
        local allsd : dir "`subdir'" dirs "`subsubd'", respectcase
        local allsd : list sort allsd
        local nsubd : word count `allsd'

        if `nsubd' == 0 {
            local nulliste `"`nulliste' "`dir1'""'
        }

        if `nsubd' > 1 {
            local dupliste  `"`dupliste' "`dir1'""'
        }
    }
    if `u'strlen(`"`nulliste'"') > 0  {
        local abbruch = 1
        local message1  "    " _n ///
        "There is no subdirectory `subsubd' in the following SUF-directories: "   _n ///
        `" `nulliste' "'  _n ///
        `"The do-file stops. Please correct the macro  "subsubd" in Section B) "'   _n ///
        "of the parameter definition or check the structure of the  "    _n ///
        "Source directories."    _n ///
        "    "
    }
    if `u'strlen(`"`dupliste'"') > 0  {
        local abbruch = 1
        local message2  "    " _n ///
        "The subdirectory `subsubd' is not unique in the following directories:"   _n ///
        `" `dupliste' "'  _n ///
        `"The do-file stops. Please correct the macro  "subsubd" in Section B) "'   _n ///
        "of the parameter definition or check the structure of the  "    _n ///
        "Source directories."    _n ///
```

```
                        "    "
                    }
                }
            }

            if `u'strlen(`"`alld'"') == 0 {    /* no SUF-Directories in rootdir */
                local abbruch = 1
                local message3  "     " _n ///
                "The subdirectories `subd' in directory `rootdir' have not been found. " _n ///
                    `"The do-file stops. Please correct the macro  "subd" in Section B) "'   _n ///
                    "of the parameter definition "    _n ///
                        "    "
            }
        }
        if _rc != 0{       /* rootdir has not been found */
            local abbruch = 1
            local message "    " _n ///
            "You did not specify a valid rootdir-name.  " _n ///
            `"The do-file stops. Please correct the macro  "rootdir" in Section B) "'   _n ///
            "of the parameter definition "    _n ///
                "    "
        }

    } /* End of loop: does rootdir exist? */

    if `u'strlen("`rootdir'") == 0{
            local abbruch = 1
            local message5 "    " _n ///
            "You did not specify a rootdir-name.  " _n ///
            `"The do-file stops. Please correct the macro "rootdir" in Section B) "'   _n ///
            "of the parameter definition "    _n ///
                "    "
    }

}   /* End of loop:  if switch == 1 */

if `u'strlen(`"`message'"') > 0       {
    display as err  " `message'"
}
if `u'strlen(`"`message1'"') > 0  {
```

```
    display as err  " `message1'"
}
if `u'strlen(`"`message2'"') > 0  {
    display as err  " `message2'"
}
if `u'strlen(`"`message3'"') > 0  {
    display as err  " `message3'"
}
if `u'strlen(`"`message4'"') > 0  {
    display as err  " `message4'"
}

if `u'strlen(`"`message5'"') > 0  {
    display as err  " `message5'"
}
if `u'strlen(`"`message6'"') > 0  {
    display as err  " `message6'"
}

if `abbruch' == 1  {
    local message ""
    local message1 ""
    local message2 ""
    local message3 ""
    local message4 ""
    local message5 ""
    local message6 ""
    exit
}

capture log close
log using "`pfad1o'/`lg'", replace

local k1 = `nrd'
forvalues l1 = 1(1)`k1'    {   /* loop over the SUF-Directories */
    if `switch' == 1       {
        local dir1 : word `l1' of `alld'
        local data "`rootdir'/`dir1'/`subsubd'"
        /* in case subsubd is empty, the last Slash has to be removed */
        local str = `u'strreverse("`data'")
```

```
    if `u'strpos("`str'", "/") == 1 {
        local str = `u'subinstr("`str'", "/", "", 1)
        local data = `u'strreverse("`str'")
    }
}

local infile : dir "`data'" files `"`files'"', respectcase
local a : word count `infile'

local infile = `u'subinstr(`"`infile'"',`"""',"",`a'*2)  /* remove the quotes */
local infile = `u'subinstr(`"`infile'"',".dta","",`a') /* remove the extension .dta */

local message "    " _n ///
           "`dir1' - The Stata Files in directory `data'  "   _n ///
       `"   matching the wildcard `files' are included.   "'   _n ///
           "    "
noisily display " `message' "

local nds : word count `infile'
local a = `nds'
local message "`a' data files will be processed"  _n ///
           "    "
noisily display " `message' "

local svd ""   /* updated file list, in case a file from the infile-list can not be opened */
local nvrs = 0  /* macro for the nVars of a file when processed in portions */
local nvrsnm = 0 /* macro for the nonmissing nVars of a file when processed in portions */

if `a' > 0 {
    forvalues i = 1(1)`a'  {       /*  loop over the files */
        local datei = word(`"`infile'"', `i')
        use  `"`data'/`datei'"', clear

        capture quietly describe, varlist
        if _rc == 601  {
            local fnexist "`fnexist' `datei'"  /* List of files not found */
        }
        if _rc == 610 {
            local s14file "`s14file' `datei'"  /* List of files not to be opened */
        }
```

```
if _rc == 0{  /* If a file is found */
    local allvars = r(varlist)
    local nsik = r(k)
    local n = `nsik'
    local message "Processing file `i', file name: `datei'.dta with `n' Vars at $S_TIME"
    noisily display "`message'"

    /* Determination of the wave indicators - all of them are identified here, because they have to
        be loaded for each portion.
        Note: the determination of the relevant wave indicator is not necessary for NEPS data, but
            the function is kept to provide for possibly more than one wave indicators in future. */

    if `u'strlen("`syrlist'") > 0 {
        local syrl ""
        local ns : word count `syrlist' /* Priority list of wave indicators according to section B)
                                           of the parameter definition */

        /* Compiling the list of actually present wave variables in allvars */
        forvalues j = 1(1)`ns' {   /* Loop: priority list of wave indicators */
            local syr = word(`"`syrlist'"', `j')
            forvalues l = 1(1)`n'  {   /* Loop over variables in file at hand */
                local av = word(`"`allvars'"', `l')
                if "`av'" == "`syr'"  {
                    local syrl "`syrl' `syr'"
                }
            }
        }
        local nsyrl : word count `syrl'  /* the list of actually present wave indicators */
    }

    /* data signature of the file */

    capture quietly datasignature report
    local chgd = r(datetime)
    if "`chgd'" == "."     {
        local infods "no data signature was previously set"
    }
    else    {
        if  r(changed) == 0 {
            local chgd = r(datetime)
```

```
        local x : display %tcDDmonCCYY_HH:MM `chgd'
        local infods "data unchanged since `x'"
    }
    if  r(changed) != 0 {
        local chgd = r(datetime)
        local x : display %tcDDmonCCYY_HH:MM `chgd'
        local infods "data has changed since `x'"
    }
}

/* Disassemble big files into portions */

local n = `nsik'
if `n' > `port'{
    local list ""
    local p = int(`n'/`port')
    local x = `p' - 1
    local span = `port'
    local beg = -`port' + 1
    forvalues i = 1(1)`x'  {
        local p1 = `beg' + `span'
        local p2 = `p1' + `span' - 1
        local w1 = word(`"`allvars'"', `p1')
        local w2 = word(`"`allvars'"', `p2')
        local list "`list' `w1'-`w2'"
        local beg = `p1'
    }
    local p1 = `beg' + `span'
    local p2 = `n'
    local w1 = word(`"`allvars'"', `p1')
    local w2 = word(`"`allvars'"', `p2')
    local list "`list' `w1'-`w2'"
}
if `n' <= `port'   {
    local w1 = word(`"`allvars'"', 1)
    local w2 = word(`"`allvars'"', `n')
    local list "`w1'-`w2'"
}
local zn : word count `list'
local z = `zn'
```

```
forvalues s = 1(1)`z'  {          /* Loop over every portion */
    local nvrs = 0  /* macro for the nVars of the file at hand (processed in portions) */
    local nvrsnm = 0 /* macro for the nonmissing nVars of the file at hand (processed in portions) */

    local p = word(`"`list'"', `s')
    if `z' == 1 {          /* only 1 portion */
        local fileop `"capture use `p' using "`data'/`datei'.dta", clear"'
        `fileop'
    }
    if `z' > 1  {          /* more than 1 portions */
        local fileop `"capture use `p' `syrl' using "`data'/`datei'.dta", clear"'
        `fileop'
    }

    if c(N) == 0 {    /* file has no cases */
        if `u'strpos("`nobsfile'", " `datei' ") == 0    {
            local nobsfile "`nobsfile' `datei' " /* list of files with 0 cases */
        }
    }
    if c(N) > 0 {  /* file has cases */
        if `s' == `z'  {   /* in the last portion */
            local svd = `u'strtrim("`svd' `datei'")
        }
        if `z' > 1 {
            local message "     Processing `datei'.dta, portion `s' of `z': `p' at $S_TIME"
            noisily display "`message'"
        }


        /* update variable list, determine no of entries, set loop counter */
        quietly describe, varlist
        local allvars = r(varlist)
        local nvars : word count `allvars'  /* r(k) should have the same result, but I count the
                                               entries in allvars to be on the safe side */
        local n = `nvars'
```

```
                    /*######  Compile the lists of the extracted information:                          ###########
                    ########  Variable related information go into lists with 1 entry per variable     ###########
                    ########    generated by looping over the variables.                               ###########
                    ########  File related infomation are compiled outside of the loop over the        ###########
                    ########    variables.
    #########*/

                     /* the following lists have to be emptied before the next file is processed    */
                     local dtyp ""
                     local lmin ""
                     local lmax ""
                     local uval ""
                     local uvalp ""
                     local valu ""
                     local valup ""
                     local nzero ""
                     local vmax ""
                     local vmin ""
                     local vminp ""
                     local smiss ""
                     local nmiss ""
                     local nsvys ""
                     local symin ""
                     local symax ""
                     local sylev ""
                     local undc ""
                     local vlab ""
                     local vlabmx ""
                     local values ""
                     local labels ""
                     local chqde ""
                     local chqen ""
                     local chfil ""
                     local chaf ""
                     local chqno ""
                     local chin ""

                     /* Number of vars with bonmiss-values, is counted up over the vars, therefore has to be set
                            to 0 before a new file is processed. */
                     local nvn = 0
```

```
/* Determination: which is the wave indicator? The procedure moves through the priority list
    from last entry to first. 'syv' is overwritten when a variable from syrlist has values
    between 0 and .
    Hence the content of syv at the end of the procedure is the name of the wave indicator
    with the highest priority.
*/

if `u'strlen("`syrlist'") > 0 {
    local syv ""
    local ns : word count `syrlist' /* priority list of the wave indicator, according to
                                       parameter definition */

    forvalues j = `ns'(-1)1    {
        local syr = word(`"`syrlist'"', `j')
        capture confirm variable `syr', exact
        if _rc == 0    {       /* `syr' exists */
            quietly summarize `syr', meanonly
            local min = r(min)
            local max = r(max)
            if r(min) < . & r(max) > 0 {   /* `syr' has a value between 0 and .  */
                local syv "`syr'"
            }
        }
    }
}

/* ########   Begin of the routines relating to variables: extract indicators and   ########
   ########        assign them to macro lists                                      ########*/

forvalues k = 1(1)`n'  {
    /* Empty all variable-related macros which fill ists or count list entries    */
    local vl ""   /* Vallabelset */
    local vlmx "" /* highest labeled value */
    local uv ""    /* Number of values */
    local uvp ""   /* Number of nonmissing values */
    local ln ""    /* Minimum length of Stringvar */
    local lx ""    /* Maximum length of Stringvar  */
    local sms ""   /* Number of system-missings */
    local nms ""   /* Number of non-missing observations */
```

```
local sys ""    /* Number of waves */
local syn ""    /* Earliest wave */
local syx ""    /* Latest wave */
local vu ""     /* Value if there is only 1 value */
local vup ""    /* nonmissing value, if there is only 1 nonmissing value */
local nz ""     /* Frequency of the value 0 */
local ud ""     /* undocumented value */
local vx ""     /* Maximum value */
local vn ""     /* Minimum value */
local vp ""     /* smallest nonmissing value */
local sylev ""  /* Waves wehre `var' has nonmissing values */
local chqde ""    /* Question text from Characteristics (German) */
local chqen ""    /* Question text from Characteristics (English) */
local chfil ""   /* Outputfilter from Characteristics  (de=en) */
local chaf ""    /* Autofill-Instruction from Characteristics (de=en) */
local chqno ""   /* Question-Number from Characteristics */
local chin ""    /* Questionnaire-Varname from Characteristics */


/* Take the first or next variable, respectively */
local var = word("`allvars'", `k')

/* Determine the question text of the k-th variable from the Characteristics */
local chqde : char `var'[NEPS_questiontext_de]
if `"`chqde'"' == ""   {
    local chqde "***"
}
local chqen : char `var'[NEPS_questiontext_en]
if `"`chqen'"' == ""   {
    local chqen "***"
}

/* Determine the outputfilter of the k-th variable from the Characteristics */
local chfil : char `var'[NEPS_outputfilter_de]
if `"`chfil'"' == ""   {
    local chfil : char `var'[NEPS_outputfilter_en]
}
if `"`chfil'"' == ""   {
    local chfil "***"
}
```

```
/* Determine the Autofillinstruction of the k-th variable from the Characteristics */
local chaf : char `var'[NEPS_autofillinstruction_de]
if `"`chaf'"' == ""    {
    local chaf : char `var'[NEPS_autofillinstruction_en]
}
if `"`chaf'"' == ""    {
    local chaf "***"
}

/* Determine the question number of the k-th variable from the Characteristics*/
local chqno : char `var'[NEPS_questionnumber]
if `"`chqno'"' == ""   {
    local chqno "***"
}

/* Determine the name in the questionnaire of the k-th var. from the Characteristics */
local chin : char `var'[NEPS_alias]
if `"`chin'"' == ""    {
    local chin "***"
}

/* Determine the data type of the k-th variable  */
local dt : type `var'

/* Name of the assigned value label Set and highest labeled value */
local vlx : value label `var'

if "`vlx'" == ""   {   /* if no value label set ist assigned to var */
    local vl "-200"
    local vlmx "-200"
}
if "`vlx'" != ""   {   /* if a value label set ist assigned to var... */
    quietly capture label list `vlx'

    if _rc > 0     {   /* ... but the assigned label set does not exist */
        local vl "-200"
        local vlmx "-200"
    }
    if _rc == 0    {   /* ... and the assigned label set exists */
```

```
        local vl `"`vlx'"'
        local vlmx = r(max)
    }
}

/* Number of values of the variable
    --> r(N_unique) of command inspect gives that too, but only up to 99 unique values
    hence not to be used here */
sort `var'
capture drop h1
by `var': gen byte h1 = cond(_n==1, 1, 0)
quietly summarize h1, meanonly
local uv = r(sum)


/* Generate the list of missing values  */
numlist "`nepsmissings'", integer sort
local misslist = r(numlist)


/* ###### Indicators applying to string variables        ###### */

if  `u'strpos("`dt'", "str") > 0   {

    /* max and min-length */
    capture drop h3
    gen h3 = `u'strlen(`var')
    quietly summarize h3 if h3 > 0, meanonly
    local ln = r(min)
    local lx = r(max)
    if `lx' == .   {
        local ln = 0
        local lx = 0
    }

    /* Auxiliary variable h2 shows if variable is empty, has a missing code,
           or contains a text */
    capture drop h2
    gen h2 = cond(`u'strlen(`u'strtrim(`var')) > 0, 1, 0)
    local v : word count `misslist'
```

```
    forvalues d = 1(1)`v'  {
        local m : word `d' of `misslist'
        quietly replace h2 = -1 if `u'strpos(`var', "`m'") > 0
    }

    /* Number of non-missings (not empty and not a missing-code) and of empty strings */
    quietly inspect h2
    local nms = r(N_pos)
    local nz = r(N_0)

/* Number of non-missing levels (file is still sorted by `var'), using the
     previeously created auxiliary variable h2
     (h2 == 1 if Stringvar is neither empty nor a missing code)
     The auxiliary variable h3 indicates a new category */

    capture drop h3
    by `var': gen byte h3 = cond(h2 == 1 & _n==1, 1, 0)
    quietly replace h3 = 0 if `var'  == ""  /* otherwise "empty" counts as a level */
    quietly summarize h3, meanonly
    local uvp = r(sum)

    /* Generating wave indicators - using the previously created auxiliary variable h2  */

    if `u'strlen("`syv'") > 0  {         /* there is a wave variable in the source data */
        quietly inspect `syv' if `syv' > 0 & `syv' <= . & h2 == 1
        local sys = r(N_unique)
        quietly summarize `syv' if h2 == 1
        local syn = r(min)
        local syx = r(max)
        if `sys' == 0  {
            local syn = -9
            local syx = -9
        }
        quietly levelsof `syv' if h2 == 1 , missing local(sylev)
    }
    if `u'strlen("`syv'") == 0 {  /* if there is no wave variable in the source data */
        local sys = -1
        local syn = -1
        local syx = -1
    }
```

```
        /* List for the value label variables */
        quietly levelsof `var' if h2 == 1 & `uvp' < 101 , clean separate(***) local(labels)
        if `u'strlen(`"`labels'"') > 0 {
            local labels = `"`labels'***"'   /* using *** as a delimiter, as yet 3 asterixes
                                                are not part of the label strings */
        }

        /* assigning defaults to the indicators which do not apply for string vars */

        local vu = -200
        local vp = -200
        local vup = -200
        local ud = -200
        local vx = -200
        local vn = -200
        local vlmx = -200
        local sms = -200


    }       /* end of the rouutines for stringvars */


    /* ###### Indicators applying for numvars (= not a stringvar) ###### */

    /* defaults for max-min-length, and identify the uniqe value, if `var' has one
            (file is still sorted by `var') */

    if  `u'strpos("`dt'", "str") == 0   {
        local ln = -200
        local lx = -200
        if `uv' == 1    {   /* uv contains the number of levels or different
                               values, respectively */
            local vu = `var'[1]
        }
        if `uv' != 1    {
            local vu = -900
        }
```

```
/* Auxiliary variable h2 shows if variable is empty, has a missing code,
       or contains another value */
capture drop h2
gen h2 = cond(`var' < ., 1, -2)
quietly replace h2 = 0 if `var' == 0
local v : word count `misslist'
forvalues d = 1(1)`v'  {
    local m : word `d' of `misslist'
    quietly replace h2 = -1 if `var' == real("`m'")
}
/* -> h2 = -2 (var == sysmis), -1 (var == missing), 0 (var == 0),
       1 (var > 0 & not missing/sysmis) */


/* Number of different nonmissing values   (file is still sorted by `var')*/
capture drop h3
by `var': gen byte h3 = cond(h2 >= 0 & _n==1, 1, 0)
quietly summarize h3, meanonly
local uvp = r(sum)

/* nonmissing value, if there is only 1 nonmissing value */
if `uvp' == 1  {
    gsort -h3      /* ATTENTION: file is being resorted */
    local vup = `var'[1]
}
if `uvp' != 1  {
    local vup = -900
}

/* Frequencies of the value 0, of nonmis values and of unlabeled values */
quietly inspect h2
local nms = r(N_pos) + r(N_0)
local nz = r(N_0)

quietly inspect `var'
local ud = r(N_undoc)
if `ud' == .   {
    local ud = 0
}
```

```
/* Frequencies of System-Missings
ATTENTION pitfall: if there are no system missings, the return codes are . */
quietly misstable summarize `var'
local sms = r(N_gt_dot) + r(N_eq_dot)
if `sms' == .  {
    local sms = 0
}

/* Maximum and minimum non-missing value of var */
quietly summarize `var', meanonly
local vx = round(r(max), .001) /* rounded, for not to exeed max length of macros  */
local vn = round(r(min), .001) /* rounded, for not to exeed max length of macros  */

/* Minimum positive value of var  */
quietly summarize `var' if h2 >= 0, meanonly
local vp = round(r(min), .001) /* rounded, for not to exeed max length of macros  */
if `vp' == .   {          /* -> there are no positive values of var */
    local vp = -900
}

/* ##### Variable-related wave indicators:                                  #########

    Number of waves plus earliest and latest wave where var is non-missing,
    using the previously generated auxiliary variable h2 */

if `u'strlen("`syv'") > 0  {    /* Wave information is present in source file */
    quietly inspect `syv' if `syv' > 0 & `syv' < . & h2 >= 0
    local sys = r(N_unique)
    quietly summarize `syv' if `syv' > 0 & `syv' < . & h2 >= 0, meanonly
    local syn = r(min)
    local syx = r(max)
    if `sys' == 0  {
        local syn = -9
        local syx = -9
    }
}

if `u'strlen("`syv'") == 0 {  /* No wave information is present in source file  */
    local sys -1
    local syn -1
```

```
        local syx -1
    }


    /*#######  Generating the wave-, values-, and value label variables  ###############
    #########               fill local lists for each of these           ##############*/


    /* ##########   Generate the wave-flag-variables                      ##############*/

    /* lists of the waves where var has nonmissing values. The option "missing"
        causes that also the level "." of syv is included in the list
        which is required to create an array structure of all the lists*/

    if `u'strlen("`syv'") > 0  {
        quietly levelsof `syv' if h2 >= 0 , missing local(sylev)
    }


    /*########   Generating the lists for nonmissing levels       ######################
    ##########       and value labels of var (max 100 levels)     ######################*/

    quietly levelsof `var' if h2 >= 0 & `uvp' < 101, local(values)
    local c : word count `values'
    if `c' > 0  {
        if "`vlx'" != "-2" {       /* if a label set has been assigned to var... */
            local labels ""
            forvalues j=1(1)`c'     {
                local wx : word `j' of `values'
                local lab : label (`var') `wx', strict
                local lab `"`lab'***"'  /* using *** as a delimiter, as yet 3 asterixes
                                           are not part of the label strings */
                if `"`lab'"' == "***"    {
                    local lab "###***"
                }
                local labels `"`labels'`lab'"'
            }
        }
    }
}   /* end of the routines for numvars */
```

```
/* Fill the lists with values from all variables  */
local uval "`uval' `uv'"   /* Number of (unique) levels */
local uvalp "`uvalp' `uvp'"    /* Number of (unique) levels >= 0 */
local lmin "`lmin' `ln'"   /* min-length (string vars) */
local lmax "`lmax' `lx'"   /* max-length (string vars) */
local valu "`valu' `vu'"   /* value of var, if there is only 1 level */
local valup "`valup' `vup'"    /* positive value of var if there is only 1 posit. level*/
local nzero "`nzero' `nz'"/* frequency of value 0 */
local vmax "`vmax' `vx'"   /* maximum of var */
local vmin "`vmin' `vn'"   /* minimum of var */
local vminp "`vminp' `vp'"/* minimum value >=0 (limited to 2 decimal places) */
local nmiss "`nmiss' `nms'"    /* number of non-missing observations   */
local smiss "`smiss' `sms'"    /* number of system-missing observations   */
local nsvys "`nsvys' `sys'"  /* number of waves where var has been surveyed */
local symin "`symin' `syn'" /* earliest wave where var has been surveyed  */
local symax "`symax' `syx'" /* latest wave where var has been surveyed  */
local undc "`undc' `ud'"   /* number of unlabeled obs. (only vars with label set) */
local vlab "`vlab' `vl'"   /* name of attached value label set */
local vlabmx "`vlabmx' `vlmx'"/* highest labeled value */


/* The following lists are overwritten with every new file, lest they might use too much
    of the working space.
    This is achieved by naming the lists alike except for an variable enumerator */

/* locals for the question texts  */
local charqde`k' `"`chqde'"'       /* German question text of k-th variable */
local charqen`k' `"`chqen'"'       /* English question text of k-th variable */

/* locals for further characteristics of the k-th variable */
local charfil`k' `"`chfil'"'
local chautf`k' `"`chaf'"'
local chquesn`k' `"`chqno'"'
local chinam`k' `"`chin'"'

/* standardized wave lists  */
local sytemp ""    /* standardized wave list of the k-th variable */
if "`sylev'" != "" {   /* previously compiled list of waves where var >= 0 */
    local c : word count `sylev'
```

```
            forvalues j = 1(1)`c'  {
                local sla : word `j' of `sylev'
                local slan = real("`sla'")

                /* unadmissible values are filtered out further down and written to output */
                local sytemp "`sytemp' `slan'" /* compilation of the list of the standardized
                                                  survey years */
            }
            local sylev`k' "`sytemp'"
        }

        /* k-th list, using the previously compiled value list (only vars with 1-100
            nonmissing levels) */
        local c : word count `values'
        if `c' > 0  {
            local values`k' "`values'"
        }

        /* k-th list, using the previously compiled value label list (only vars with 1-100
            nonmissing levels) */
        if `u'strlen(`"`labels'"') > 0{
            local labels`k' `"`labels'"'
        }

        /* update the enumerator for vars with nonmissing observations */
        if `nms' > 0    {
            local nvn = `nvn' + 1
        }
    }   /* End loop over the variables of a file */


/* #### End routines for variables / begin routines for the entire file or portion  ######*/
capture drop h1
capture drop h2
capture drop h3


/* No. of obs, no. of vars and no. of nonmissing vars of a file */
local no = c(N)/* no. of obs. */
local nv = c(k)/* no. of vars */
```

```
/* no. of vars with nonmiss obs has been counted up above, in macro nvn within the loop
    over the variables */


/* ###### Generating the metafile for the file just being processed       ################*/

/* Create a basic metafile: only variable names and variable labels
   in different language versions, if there are any  */
quietly label language
local nl = r(k)
local langs = r(languages)
local currl = r(language)
local langsoth ""  /* List of the other than current languages */
forvalues nn = 1(1)`nl'    {
    local lanx : word `nn' of `langs'
    if "`lanx'" != "`currl'"   {
        label language `lanx'
        describe, replace clear /* this command creates the basic metafile */
        if c(N) > 0    {  /* if the new file has records */
            keep name varlab
            rename name varname
            rename varlab varlab_`lanx'
            save "`pfad1d'/temp_`lanx'.dta", replace
            local langsoth "`langsoth' `lanx'"
        }
    }

    `fileop'    /* reopen the source file just being processed */
}
label language `currl'

/* create a 2nd basic metafile: var name, var label, var type, isnumeric, and merge the
   1st basic metafile which contains the var labels in different languages (if available) */
describe, replace clear
if c(N) > 0    {  /* if the new file has records */
    keep name varlab type isnumeric
    rename name varname
    rename varlab varlab_`currl'
    /* merge metafile 1 with varlabels in different languages */
    if `u'strlen("`langsoth'") > 0{
```

```
    local nlango : word count `langsoth'
    forvalues o = 1(1)`nlango'     {
        local lango : word `o' of `langsoth'
        merge 1:1 varname using "`pfad1d'/temp_`lango'.dta"
        capture erase "`pfad1d'/temp_`lango'.dta"
    }
}
capture drop _merge


/* #### Add file related information #### */
/* Source file name */
capture drop datenfile
gen str datenfile = "`datei'"
lab var datenfile "Source file"
order datenfile, first
/* Data signature */
capture drop dsign
gen str40 dsign  = "`infods'"
lab var dsign "Data signature (date)"
order dsign, after(datenfile)
/* No. of observations in source file */
capture drop nobs
gen nobs = `no'
lab var nobs "No. of observations in source file"
order nobs, after(dsign)
/* No. of variables in source file */
capture drop nvrs
gen nvrs = `nv'
lab var nvrs "No. of vars in source file"
order nvrs, after(nobs)
/* No. of variables with non-missing values in source file*/
capture drop nvrsnm
gen nvrsnm = `nvn'
lab var nvrsnm "No. of vars with non-miss values in source file"
order nvrsnm, after(nvrs)
```

```
/* #### Add variable related information #### */
/* No. of non-missing observations in variable */
capture drop nonmis
gen long nonmis = 0
lab var nonmis "No. of non-miss obs in var"
/* No. of system-missing observations in variable  */
capture drop sysmis
gen long sysmis = 0
lab var sysmis "No. of system-miss obs in var"
/* No. of levels in variable */
capture drop uniqvals
gen uniqvals = 0
lab var uniqvals "No. of levels in var"
/* No. of levels in var >= 0 (namely nonmissing) */
capture drop uniqvalpos
gen uniqvalpos = 0
lab var uniqvalpos "No. of levels in var >= 0"
/* Value, if there is only 1 level in variable */
capture drop valuniq
gen long valuniq = .
lab var valuniq "Value if only 1 level"
/* Non-missing value, if there is only 1 non-missing level */
capture drop valuniqpos
gen long valuniqpos = .
lab var valuniqpos "Value if only 1 level >= 0"
/* Frequency of value 0 in numVar */
capture drop nvalzero
gen long nvalzero = .
lab var nvalzero "Frequency of value 0 "
/* Minimum value of the variable */
capture drop valmin
gen long valmin = .
lab var valmin "Minimum value of var"
/* Minimum value >= 0, namely minimum nonmissing value */
capture drop valminp
gen long valminp = .
lab var valminp "Minimum value of var >= 0"
/* Maximum value of the variable */
capture drop valmax
gen long valmax = .
```

```
lab var valmax "Maximum value of var"
/* Name of the attached value label set */
capture drop vallabset
gen str vallabset = ""
lab var vallabset "Name of value label set"
/* Number of unlabeled observations (with labeled variables only */
capture drop undoc
gen long undoc = .
lab var undoc "No. of unlabeled obs (labeled vars only)"
/* Highest labeled value */
capture drop vlabmax
gen vlabmax = .
lab var vlabmax "Maximum labeled value"
/* Minimal length of string variable */
capture drop strmin
gen long strmin = .
lab var strmin "Min. length of string var"
/* Maximal length of string variable  */
capture drop strmax
gen long strmax = .
lab var strmax "Max. length of string var"

/* Vars for the wave indicators */
capture drop wind
gen str wind = "`syv'"
if `u'strlen("`syv'") == 0 {
    quietly replace wind = "-1"
}
lab var wind "Wave indicator"
capture drop nwaves
gen byte nwaves = 0
lab var nwaves "No. of waves"
capture drop wavemin
gen wavemin = 0
lab var wavemin "Earliest wave"
capture drop wavemax
gen wavemax = 0
lab var wavemax "Latest wave"
```

```
/* Blanco-Variables for the waves */
local n = `nw'
forvalues j = 1(1)`n'  {
    local scrpt = `j'-1+`wave1'
    capture drop wav`scrpt'
    gen int wav`scrpt' = 0
}

/* Variables for the question texts */
capture drop questde
gen str questde = ""
lab var questde "German question text"
capture drop questen
gen str questen = ""
lab var questen "English question text"

/* Variable for the Outputfilter */
capture drop ofilter
gen str ofilter = ""
lab var ofilter "Output filter"

/* Variable for the Autofill-Instruction */
capture drop afill
gen str afill = ""
lab var afill "Autofill instruction"

/* Variable for the Question number in the questionnaire */
capture drop iquestno
gen str iquestno = ""
lab var iquestno "Question no. in questionnaire"

/* Variable for the variable name in the questionnaire */
capture drop ivarname
gen str ivarname = ""
lab var ivarname "Varname in questionnaire"

/* Blanco-variables for the values (only varables with 1-100 levels) */
forvalues j = 1(1)100  {
    local scrpt = `j'
    capture drop val`scrpt'
```

```
    gen val`scrpt' = .
}

/* Blanco-variablen for the labels (only varables with 1-100 levels) */
forvalues j = 1(1)100  {
    local scrpt = `j'
    capture drop lab`scrpt'
    gen str lab`scrpt' = ""
}

/* Note: the variables undoc as well as val# and lab# will be modified further down, after
    compiling the single metafiles */

local nok " "  /* list for unadmissible values of the wave indicator (--> to Output*/
local n  = `nvars'
forvalues k = 1(1)`n'  {
    local var : word `k' of `allvars'
    local uv : word `k' of `uval'
    local uvp : word `k' of `uvalp'
    local vu : word `k' of `valu'
    local vup : word `k' of `valup'
    local nz : word `k' of `nzero'
    local vx : word `k' of `vmax'
    local vn : word `k' of `vmin'
    local vp : word `k' of `vminp'
    local lx : word `k' of `lmax'
    local ln : word `k' of `lmin'
    local nms : word `k' of `nmiss'
    local sms : word `k' of `smiss'
    local sys : word `k' of `nsvys'
    local syn : word `k' of `symin'
    local syx : word `k' of `symax'
    local ud : word `k' of `undc'
    local vl : word `k' of `vlab'
    local vlmx : word `k' of `vlabmx'

    quietly {
        replace nonmis = `nms' if varname == "`var'"
        replace sysmis = `sms' if varname == "`var'"
        replace uniqvals = `uv' if varname == "`var'"
```

```
replace uniqvalpos = `uvp' if varname == "`var'"
replace valuniq = `vu' if varname == "`var'"
replace valuniqpos = `vup' if varname == "`var'"
replace nvalzero = `nz' if varname == "`var'"
replace valmax = `vx' if varname == "`var'"
replace valmin = `vn' if varname == "`var'"
replace valminp = `vp' if varname == "`var'"
replace strmax = `lx' if varname == "`var'"
replace strmin = `ln' if varname == "`var'"
replace nwaves = `sys' if varname == "`var'"
replace wavemin = `syn' if varname == "`var'"
replace wavemax = `syx' if varname == "`var'"
replace undoc = `ud' if varname == "`var'"
replace vallabset = "`vl'" if varname == "`var'"
replace vlabmax = `vlmx' if varname == "`var'"


/* Fill the wav# variables */

if `u'strlen("`syv'") == 0  {    /* if there is no wave indic. in source file */
    local mark 0
    local c = `nw'
    forvalues j = 1(1)`c'   {
        local scrpt = `j'-1+`wave1'
        replace wav`scrpt' = -1
    }
}

if `u'strlen("`syv'") > 0   {    /* if there is a wave indic. in source file */
    local c : word count `sylev`k''
    forvalues j = 1(1)`c'   {
        local slan = word("`sylev`k''", `j')
        local mark = real("`slan'")
        if `mark' >= `wave1' & `mark' <= `wave2'    {
            capture confirm new variable wav`slan', exact
            if _rc != 0{
                replace wav`slan' = 1 if varname == "`var'"
            }
        }
        if `mark' < `wave1' | `mark' > `wave2' {
```

```
            if `u'strpos(" `nok' ", " `mark' ") == 0     {
                local nok "`nok' `mark'"
            }
        }
    }
    local sylev`k' ""  /* mandatory, lest its content might be transferred
                          to the next source file in certain cases */
}

/* Fill the value variables */
if `u'strlen("`values`k''") > 0  {
    local c : word count `values`k''
    forvalues j = 1(1)`c'   {
        local val = word("`values`k''", `j')
        local valn = real("`val'")
        capture confirm new variable val`j', exact
        if _rc != 0    {
            replace val`j' = `valn' if varname == "`var'"
        }
    }
    local values`k' "" /* mandatory, lest its content might be transferred
                          to the next source file in certain cases */
}

/* Fill the value label variables */
if `u'strlen(`"`labels`k''"') > 0  {
    local str = `"`labels`k''"'
    local a = `u'strpos(`"`str'"', "***")
    local x = 1
    while `a' > 0  {
        local lb = `u'substr(`"`str'"', 1,`a'-1)
        capture confirm new variable lab`x', exact
        if _rc != 0{
            replace lab`x' = `"`lb'"' if varname == "`var'"
        }
        local str = `u'substr(`"`str'"',`a'+3,.) /* rest of the string to end */
        local a = `u'strpos(`"`str'"', "*")
        local x = `x' + 1
    }
```

```
            local labels`k' "" /* /* mandatory, lest its content might be transferred
                                  to the next source file in certain cases */ */
        }

        /* Fill the variables for question text and outputfilter */
        replace questde = `"`charqde`k'"' if  varname == "`var'"
        replace questen = `"`charqen`k'"' if  varname == "`var'"

        replace ofilter = `"`charfil`k'"' if  varname == "`var'"
        replace afill = `"`chautf`k'"' if  varname == "`var'"
        replace iquestno = `"`chquesn`k'"' if  varname == "`var'"
        replace ivarname = `"`chinam`k'"' if  varname == "`var'"


    }  /* End of: quietly */
}   /* End of loop over the variables of a portion  */

if `u'strlen("`nok'") > 0  & "`nok'" != " " {
    local message "   " _n ///
        "The wave indicator `syv' in file `datei' "_n ///
        "   includes the possibly inadmissible value(s)  `nok'" _n ///
        "      "
    noisily display "`message'"
}

save "`pfad1d'/meta_`datei'_xxx`s'.dta", replace

if `s' == `zn'     {   /* last portion is reached: merge the portion files */
    if `zn' > 1    {
        local p = `zn'
        forvalues t = 1(1)`p'  {
            use "`pfad1d'/meta_`datei'_xxx`t'.dta", clear
            local nvrs = `nvrs' + nvrs[1]
            local nvrsnm = `nvrsnm' + nvrsnm[1]
        }
        local p = `zn'
        use "`pfad1d'/meta_`datei'_xxx1.dta", clear
        forvalues t = 2(1)`p'  {
            append using "`pfad1d'/meta_`datei'_xxx`t'.dta"
        }
```

```
                                /* subtract the number of survey variables */

                                if `nsyrl' > 0 {
                                    local abz = (`zn'-1)*`nsyrl'
                                    quietly {
                                        replace nvrs = `nvrs'-`abz'
                                        replace nvrsnm = `nvrsnm'-`abz'
                                        duplicates drop
                                    }
                                }
                            }

                            save "`pfad1d'/meta_`datei'_`we'.dta", replace

                            /* Delete the single portion files */
                            local p = `zn'
                            forvalues t = 1(1)`p'  {
                                capture erase "`pfad1d'/meta_`datei'_xxx`t'.dta"
                            }
                        }   /* End: in a last portion */
                    }       /* End: create a metafile for a source file */
                }       /* End: if the source file has observations */
            }       /* End: Processing of a file / portion */
        }       /* End: if file is found */
    }       /* End: loop over all source files  */
}    /* End: if `a' > 0 = if there are source files in the directory */


if `u'strlen("`svd'") > 0  {

    /* Assemble the single metafiles (only if there are more than 1) */
    local nds : word count `svd' /* initialize enumerator */
    local a = `nds'

    local datei = word("`svd'", 1)
    use "`pfad1d'/meta_`datei'_`we'.dta", clear
```

```
        if `a' > 1 {
            local message "    " _n ///
                        "`dir1' - The single metafiles are being assembled " _n ///
                        "    "   _n ///
                        "Processing... Please wait. "  _n ///
                        "    "
            noisily display "`message'"

            forvalues i = 2(1)`a'  {
                local datei = word("`svd'", `i')
                append using "`pfad1d'/meta_`datei'_`we'.dta"
            }
        }

        /* Add a variable for the start cohort to the meta file */
        capture drop stcohor
        gen str10 stcohor = ""
        replace stcohor = "SUF " + `u'substr(datenfile, 1, 3)
        lab var stcohor "Start cohort"
        order stcohor, before(datenfile)

        save "`pfad1d'/temp_`we'`dir1'_`datum'", replace

        /* Delete the single metafiles */
        local a = `nds'
        forvalues i = 1(1)`a' {
            local datei = word(`"`svd'"', `i')
            capture erase "`pfad1d'/meta_`datei'_`we'.dta"
        }
    }       /* End: Assemble the single files of a start cohort */
}           /* End: Loop over the directories of the start cohorts */


if `abbruch' == 1  {
    exit
}
```

```
/* Assemble the single metafiles of the start cohorts while determining the first one */

local n = `nrd'
forvalues i = `n'(-1)1 {
    local dir2 : word `i' of `alld'
    capture quietly describe using "`pfad1d'/temp_`we'`dir2'_`datum'", varlist
    if _rc == 0 {   /* file exists */
        local dir1 = "`dir2'"
        local a = `i'+1
    }
}

use "`pfad1d'/temp_`we'`dir1'_`datum'", clear
capture erase "`pfad1d'/temp_`we'`dir1'_`datum'.dta"

local n = `nrd'
if `n' > `a'   {
    local message "    " _n ///
                `"The meta files of the start cohorts `alld' are being assembled "' _n ///
                "    "   _n ///
                "Processing... Please wait. "  _n ///
                "    "
    noisily display "`message'"

    forvalues i = `a'(1)`n'    {
        local dir1 : word `i' of `alld'
        capture quietly describe using "`pfad1d'/temp_`we'`dir1'_`datum'", varlist
        if _rc == 0 {   /* file exists */
            append using "`pfad1d'/temp_`we'`dir1'_`datum'"
            capture erase "`pfad1d'/temp_`we'`dir1'_`datum'.dta"
        }
    }

}
/*##########################################################################################################
  ################## Generate variables and modifications in the overall metafile  ###################################
  ##################                                                               ###################################
  ##########################################################################################################*/
```

```
/* Modification of the variable undoc, if only the missing codes and possibly value 0 are labeled  */
capture confirm new variable undoc, exact
if _rc == 110  {   /* var exists */
    replace undoc = -900 if undoc == nonmis & undoc != .
    replace undoc = -900 if undoc == (nonmis - nvalzero) & undoc != .
}
/* Modification of the val#- und lab# vars, for string source vars, or numeric source varswith more than 100 non-
    missing levels, or with less than 100 levels, but only missing codes.
*/

forvalues x= 1(1)100   {
    quietly    {
        capture confirm new variable val`x'
        if _rc == 110  {
            replace val`x' = -200 if isnumeric == 0
            replace val`x' = -100 if isnumeric == 1 & uniqvalpos > 100
            replace val`x' = -900 if isnumeric == 1 & uniqvalpos <= 100 & nonmis == 0
        }
        capture confirm new variable lab`x'
        if _rc == 110  {
            replace lab`x' = "-200" if vallabset == "-200" & isnumeric == 1
            replace lab`x' = "-100" if isnumeric == 1 & vallabset != "-200" & uniqvalpos > 100
            replace lab`x' = "-100" if isnumeric == 0 & uniqvalpos > 100
            replace lab`x' = "-900" if isnumeric == 1 & vallabset != "-200" & uniqvalpos <= 100 & nonmis == 0
        }
    }
}

/* Modification of the variables questde and questen */
quietly    {
    replace questde = "" if questde == "***"
    replace questen = "" if questen == "***"
    replace ofilter = "" if ofilter == "***"
    replace afill = "" if afill == "***"
    replace iquestno = "" if iquestno == "***"
    replace ivarname = "" if ivarname == "***"
}
```

```
/* Include record number and variable for start cohort tag */
capture drop lfn
gen long lfn = _n
order lfn, first
lab var lfn "Record number"

/* fill start cohort, if yet empty  */
capture confirm new variable stcohor
if _rc == 110  {
    if stcohor[1] == ""    {
        replace stcohor = "SUF " + `u'substr(datenfile, 1, 3)
    }
}

/* ##################### Generate a string variable that maps the waves      ############################
                          (tested: the syntax works for at least 100 waves)
*/

capture drop waves
gen str waves = ""
lab var waves "Waves where var has nonmiss values"
order waves, after(wavemax)

if `wave1' != 0 & `wave2' >= `wave1'  { /* 0 is the default if wave1 and wave2 are empty in the parameter
                                           definition */
    local wellen "`wave1'/`wave2'"
    numlist "`wellen'", integer sort
    local wlist = r(numlist)
    local nw : word count `wlist'
    local n = `nw'
    local comm "replace waves = "
    local comm1 `"replace waves = "-1" if "'
    local rsum = 0
    forvalues i = 1(1)`n'  {
        capture drop tempwav`i'
        gen str tempwav`i' = ""
        replace tempwav`i' = strofreal(`i') + " " if wav`i' == 1
        local comm = `"`comm' tempwav`i' +"'
        local comm1 = `"`comm1' wav`i' == -1 | "'
    }
```

```
    /* take off the  "+" and "|" at the end of the string */
    local comm = substr(`"`comm'"', 1, ustrlen(`"`comm'"') - 2)
    local comm1 = substr(`"`comm1'"', 1, ustrlen(`"`comm1'"') - 3)

    /* execute the replace-commands */
    `comm'
    `comm1'

     capture drop wsum
     egen wsum = rowtotal(wav`wave1' - wav`wave2')
     replace waves = "0" if wsum == 0
     capture drop wsum

     local n = `nw'
     forvalues i = 1(1)`n' {
        capture drop tempwav`i'
     }
}

/* Changing the var sequence */

order varlab* ivarname iquestno quest* ofilter afill, after(varname)
order vlabmax, after(valmax)

label data "`reslab'"
datasignature set, reset
save "`pfad1d'/`result'", replace

local dir1 : word 1 of `alld'     /* delete the last of the tempfiles */
capture erase "`pfad1d'/temp_`we'`dir1'_`datum'.dta"

/* Generation of SC-specific metafiles */

quietly levelsof stcohor, local(stclist)  /* extract values of startcohort-variable */
local nsc : word count `stclist'

local n = `nsc'
forvalues i = 1(1)`n'  {
    preserve
        local stc : word `i' of `stclist'
```

```
        local krz =  `u'substr("`stc'", 5, `u'strlen("`stc'"))
        keep if stcohor == "`stc'"
        save "`pfad1d'/metaf_`we'`krz'_`datum'", replace

    restore
}

/*##########################################################################################################################
##############################                                                  ##############################################
##############################      DISPLAY RUNTIME OF THE PROGRAM              ##############################################
##############################                                                  ##############################################*/

quietly {
    timer off 1
    quietly timer list
    scalar `time'=(r(t1))
    local hrs = int(`time'/3600)
    scalar `time' = (r(t1)) - (`hrs' * 3600)
    local min = int(`time'/60)
    local sec = round(`time' - (`min' * 60), .01)

    if `u'strlen("`fnexist'")  == 0    {
        local fnexist "none"
    }

    if `u'strlen("`s14file'")  == 0    {
        local s14file "none"
    }
    if `u'strlen("`nobsfile'") == 0    {
        local nobsfile "none"
    }
    local m1 "The generated metafile is named:"
    local m2 "`result'"
    local m3 ""

    if `u'strlen("`u'") > 0    {
        local message "  " _n ///
        "Job completed.   " _n ///
        "Runtime of the program: `hrs' hrs `min' min `sec' sec  " _n ///
        "END at $S_TIME on $S_DATE " _n ///
```

```
        "  " _n ///
        " `m1' " _n ///
        "   `m2' "  _n ///
        "   `m3' "  _n ///
        "  in Directory `pfad1d' " _n ///
        "   "  _n ///
        " Files that could not be processed: " _n ///
        "   "  _n ///
        "....Files not found: `fnexist' " _n ///
        "   "  _n ///
        "....Files with 0 cases: `nobsfile' " _n ///
        "  "
    }
    if `u'strlen("`u'") == 0   {
        local message "  " _n ///
        "Job completed.   " _n ///
        "Runtime of the program: `hrs' hrs `min' min `sec' sec  " _n ///
        "END at $S_TIME on $S_DATE " _n ///
        "  " _n ///
        " `m1' " _n ///
        "   `m2' "  _n ///
        "   `m3' "  _n ///
        "  in Directory`pfad1d' " _n ///
        "   "  _n ///
        " Files that could not be processed: " _n ///
              "   "  _n ///
        "....Files not in the actual Stata format: `s14file'  " _n ///
              "   "  _n ///
        "....Files not found: `fnexist' " _n ///
              "   "  _n ///
        "....Files with 0 cases:  `nobsfile' " _n ///
        "  "
    }
}       /* Ende: quietly */

noisily display " `message' "  _n ///
        " `message1'"

capture log close
set output proc
```