

SOEP Survey Papers

Series G – Variable Descriptions and Coding

SOEP – The German Socio-Economic Panel study at DIW Berlin

2018

SOEP-Metafile.do – a Stata Do-file to Generate a Metafile for the SOEP Data Files

Klaudia Erhardt

Running since 1984, the German Socio-Economic Panel study (SOEP) is a wide-ranging representative longitudinal study of private households, located at the German Institute for Economic Research, DIW Berlin.

The aim of the SOEP Survey Papers Series is to thoroughly document the survey's data collection and data processing. The SOEP Survey Papers is comprised of the following series:

Series A – Survey Instruments (Erhebungsinstrumente)

Series B – Survey Reports (Methodenberichte)

Series C – Data Documentation (Datendokumentationen)

Series D – Variable Descriptions and Coding

Series E – SOEPmonitors

Series F – SOEP Newsletters

Series G – General Issues and Teaching Materials

The SOEP Survey Papers are available at <http://www.diw.de/soepsurveypapers>

Editors:

Dr. Jan Goebel, DIW Berlin

Prof. Dr. Stefan Liebig, DIW Berlin and Universität Bielefeld

Dr. David Richter, DIW Berlin

Prof. Dr. Carsten Schröder, DIW Berlin and Freie Universität Berlin

Prof. Dr. Jürgen Schupp, DIW Berlin and Freie Universität Berlin

Please cite this paper as follows:

Klaudia Erhardt. 2018. SOEP-Metafile.do – a Stata Do-file to Generate a Metafile for the SOEP Data Files. SOEP Survey Papers 561: Series G. Berlin: DIW/SOEP



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

© 2018 by SOEP

ISSN: 2193-5580 (online)

DIW Berlin

German Socio-Economic Panel (SOEP)

Mohrenstr. 58

10117 Berlin

Germany

soeppapers@diw.de

SOEP-Metafile.do – a Stata Do-file to Generate a Metafile for the SOEP Data Files

Application of the do-file and documentation of the resulting meta file

Klaudia Erhardt

SOEP-Metafile.do

Download: <https://doi.org/10.5281/zenodo.1256001>

SOEP-Metafile.do do-file to generate a metafile for the SOEP data files.

Varlabels_Metafile.do auxiliary do-file to replace German and English metafile-variable labels.

System requirements:

The do-file was developed with Stata version 13, is adapted to the Unicode-reorientation of Stata version 14, and has been tested to run with Stata version 15.

Requires Stata IC or higher.

Contact to the author for questions and feedback:

erhardtk@gmx.de

Content

1	Introduction and overview	3
2	On the utility of a SOEP-Metafile.do-generated metafile for the SOEP data files	4
3	Explanation of some basic terms	5
4	Documentation of the metafile	7
4.1	Synopsis of the metafile variables	7
4.2	Detailed documentation of the metafile-variables	8
5	How to generate a metafile on the SOEP data using SOEP-Metafile.do	13
5.1	Hints and recommendations on the use of SOEP-Metafile.do	13
5.2	Adaptations required in the syntax segment "Definition of parameters"	14
6	Syntax examples on how to use the metafile	16
6.1	Create views of the metafile in the Stata data editor	17
6.1.1	Selection of metafile-variables	17
6.1.2	Selection of observations	17
6.2	Selection of variables from the SOEP Long-files	18
6.3	The use of an auxiliary variable to mark selected variable sets	19
6.4	How to capture the selected variable names for further uses	21
6.5	Proceeding if the selected variables are located in several data files	22
7	Flow chart of SOEP-Metafile.do	24
8	Complete syntax of SOEP-Metafile.do	26

Note

The SOEP website does not yet provide a DOI for downloadable software. Therefore, for the present, SOEP-Metafile.do is offered for download via the EU-funded open-science repository Zenodo.

1 Introduction and overview

This paper presents the Stata do-file SOEP-Metafile.do (version for SOEP-Users). The do-file is used to generate a summary file (metafile) of the SOEP data. To execute SOEP-Metafile.do in their local environment, users have to make only a few adjustments to the path definitions and options. Apart from this, the generation of a metafile with SOEP-Metafile.do is completely automated.

The metafile is, for its part, also a Stata file that holds data on every variable in the SOEP data files that were included in the processing by SOEP-Metafile.do. To give an impression, the figure on the next page shows a view of the metafile in the Stata data editor.

The next section describes why the metafile that is generated by the SOEP-Metafile.do is useful for SOEP users.

Section 3 explains some basic terms that are used in this paper.

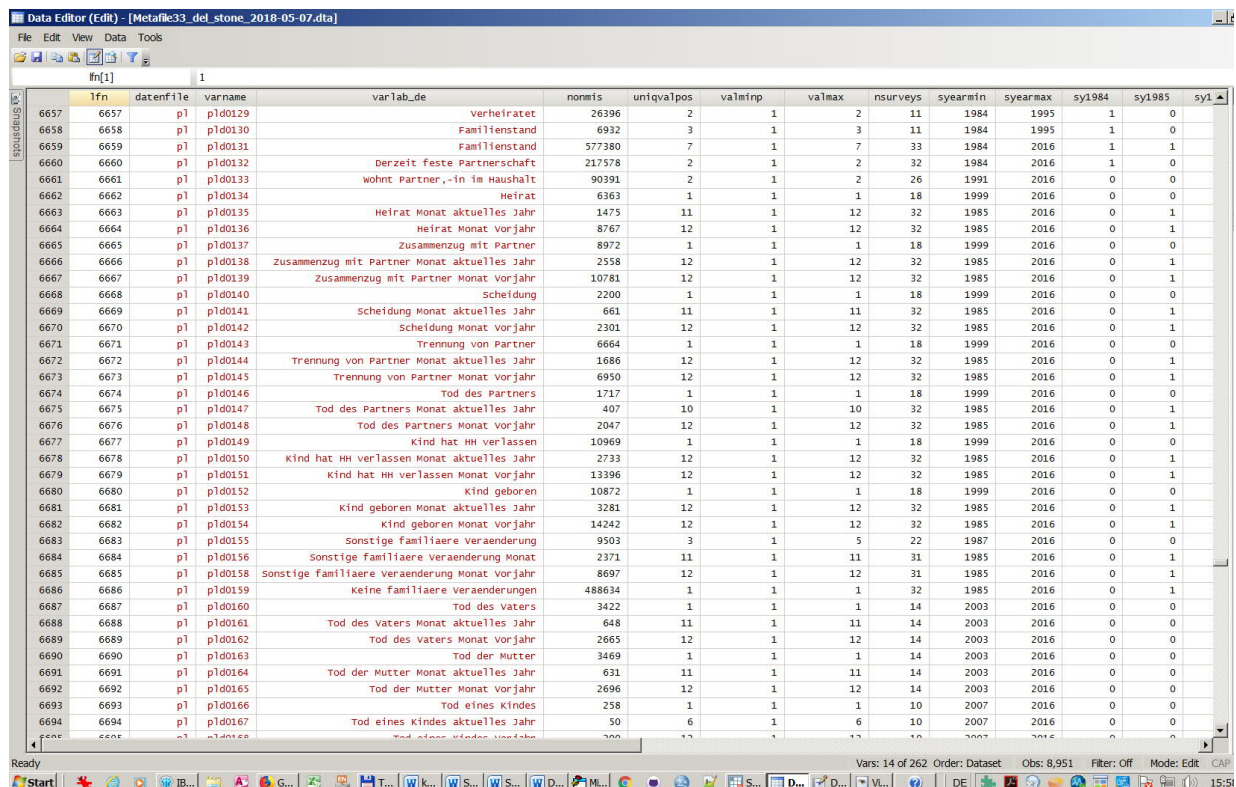
Section 4 holds the detailed documentation of the metafile and its variables.

In section 5 you find the practical instructions how to generate a metafile on the SOEP data using SOEP-Metafile.do in your local environment. It describes the choices for the parameter setting and the effects they have.

Section 6 holds examples of Stata syntax to show how you can use the metafile as a basic information system on the SOEP data.

For those who are closer interested in the programming, section 7 contains a flowchart of SOEP-Metafile.do, and section 8 holds the complete syntax of the do-file.

View of the metafile in the Stata data editor



lfn	datenfile	varname	varlab_de	nonmis	uniquvalpos	valminp	valmax	nsurveys	yearmin	yearmax	sy1984	sy1985	sy1
6657	6657	p1	p1d0129	26396	2	1	2	11	1984	1995	1	0	
6658	6658	p1	p1d0130	6932	3	1	3	11	1984	1995	1	0	
6659	6659	p1	p1d0131	577380	7	1	7	33	1984	2016	1	1	
6660	6660	p1	p1d0132	217578	2	1	2	32	1984	2016	1	0	
6661	6661	p1	p1d0133	90391	2	1	2	26	1991	2016	0	0	
6662	6662	p1	p1d0134	6363	1	1	1	18	1999	2016	0	0	
6663	6663	p1	p1d0135	1475	11	1	12	32	1985	2016	0	1	
6664	6664	p1	p1d0136	8767	12	1	12	32	1985	2016	0	1	
6665	6665	p1	p1d0137	8972	1	1	1	18	1999	2016	0	0	
6666	6666	p1	p1d0138	2558	12	1	12	32	1985	2016	0	1	
6667	6667	p1	p1d0139	10781	12	1	12	32	1985	2016	0	1	
6668	6668	p1	p1d0140	2200	1	1	1	18	1999	2016	0	0	
6669	6669	p1	p1d0141	661	11	1	11	32	1985	2016	0	1	
6670	6670	p1	p1d0142	2301	12	1	12	32	1985	2016	0	1	
6671	6671	p1	p1d0143	6664	1	1	1	18	1999	2016	0	0	
6672	6672	p1	p1d0144	1686	12	1	12	32	1985	2016	0	1	
6673	6673	p1	p1d0145	6950	12	1	12	32	1985	2016	0	1	
6674	6674	p1	p1d0146	1717	1	1	1	18	1999	2016	0	0	
6675	6675	p1	p1d0147	407	10	1	10	32	1985	2016	0	1	
6676	6676	p1	p1d0148	2047	12	1	12	32	1985	2016	0	1	
6677	6677	p1	p1d0149	10969	1	1	1	18	1999	2016	0	0	
6678	6678	p1	p1d0150	2733	12	1	12	32	1985	2016	0	1	
6679	6679	p1	p1d0151	13396	12	1	12	32	1985	2016	0	1	
6680	6680	p1	p1d0152	10872	1	1	1	18	1999	2016	0	0	
6681	6681	p1	p1d0153	3281	12	1	12	32	1985	2016	0	1	
6682	6682	p1	p1d0154	14242	12	1	12	32	1985	2016	0	1	
6683	6683	p1	p1d0155	9503	3	1	5	22	1987	2016	0	0	
6684	6684	p1	p1d0156	2371	11	1	11	31	1985	2016	0	1	
6685	6685	p1	p1d0158	8697	12	1	12	31	1985	2016	0	1	
6686	6686	p1	p1d0159	488634	1	1	1	32	1985	2016	0	1	
6687	6687	p1	p1d0160	3422	1	1	1	14	2003	2016	0	0	
6688	6688	p1	p1d0161	648	11	1	11	14	2003	2016	0	0	
6689	6689	p1	p1d0162	2665	12	1	12	14	2003	2016	0	0	
6690	6690	p1	p1d0163	3469	1	1	1	14	2003	2016	0	0	
6691	6691	p1	p1d0164	631	11	1	11	14	2003	2016	0	0	
6692	6692	p1	p1d0165	2696	12	1	12	14	2003	2016	0	0	
6693	6693	p1	p1d0166	258	1	1	1	10	2007	2016	0	0	
6694	6694	p1	p1d0167	50	6	1	6	10	2007	2016	0	0	
6695	6695	p1	p1d0168	200	12	1	12	10	2007	2016	0	0	

2 On the utility of a SOEP-Metafile.do-generated metafile for the SOEP data files

The SOEP is a highly complex panel study that is conducted yearly since 1984. Over the years, numerous different samples have been included. In wave 33 of the panel (data release 2017), the number of data files handed over to the users holding a data contract amounted to 440 wave specific files with 72,802 variables and 18 files of SOEPLong with 8,951 variables. The biggest file, pl, holds alone 3,170 variables.

The SOEP-Metafile.do-generated metafile is a useful support to orientate oneself in the complex data stock of the SOEP. It is an overall table of all the variables of those SOEP data files included in the generation of the metafile. Each variable of the metafile (i.e. column of the table) represents a certain characteristic of the SOEP data files or their variables, respectively. Each record of the metafile (i.e. line of the table) stands for a variable of a SOEP data file (see the view of the metafile in the figure above).

For example, the metafile-variables show how many usable observations a SOEP-variable has, how often and in which years it was surveyed, and more (see section 4, documentation of the metafile). So you can, for instance, identify with the aid of the metafile those variables of SOEPLong that are available for a certain time span, with at least X waves and with at least Y useable observations. The list of the selected variables can be used as a starting point for

searches in the SOEP information system, [Paneldata](#), in order to retrieve further information on the variables. Or it can be used in self-written do-files (see syntax examples in section 6).

Because the metafile is - like the source files processed to generate it - a Stata data file, all features of Stata to select and sort data and for reusing selected information from the metafile are available, be it within Stata or in other applications, like Microsoft Excel, that are able to read files in one of the export formats of Stata.

SOEP-Metafile.do is tuned to process SOEP data, but it does not need to be original SOEP data. The do-file can also be used to create a metafile for modified SOEP data that the user is working with.

Last, but not least, the metafile is a tool for those users of the SOEP data who have access only to Stata version IC and, thus, cannot open SOEP data files that have too many variables for Stata IC. In these cases, the metafile can be used as an information resource to select the variables for the Stata command use <varlist>, using <SOEP data file>, which is used to load subsets of data files to the working memory.

With the help of my comments in the syntax of section 8, those who are interested in the way SOEP-Metafile.do is programmed may follow the solutions I developed, such as to measure the runtime of the job or to segmentize big data files in order to reduce the runtime, using them as an incitation for self-written Stata programs.

3 Explanation of some basic terms

Some of the terms used in this paper are self-created, while others relate to SOEP-specific conditions. The following short explanations clarify their meaning in the context at hand.

Nonmissing values

The SOEP standard for missing values are one-digit negative integers, i.e. numerals between -1 and -9. Therefore, in the syntax of SOEP-Metafile.do, "non-missing values" are defined as values ≥ 0 & $< \text{sysmis}$ (see next paragraph). However, some SOEP-variables may have negative values that are not actually missing values. Mostly, these are generated income and wealth indicators. They do not receive a special treatment in the syntax, because there is no succinct algorithm to identify them.

Consequently, for those variables the number of non-missing observations is not reported correctly in the generated metafile. In most cases the concerned variables can be identified by the metafile-variable `valmin` (minimal value), as long as the minimal value is smaller than -9 and not equal to -200:

```
edit datenfile varname varlab valmin if valmin < -9 & valmin != -200
```

Note: -200 is a metafile-specific missing code, see section 4.2.

System missings or sysmis

Term to denominate the Stata missing codes `., .a, .b, (...), .z,` to distinguish from the SOEP-specific missing codes. The standard Stata missing code `."` is easily overlooked in the running text. Therefore, `<sysmis>` oder `sysmis` is mostly used throughout this paper.

Indicators and type 2-indicators

"Indicators" is the term used in this paper to denominate characteristics that are extracted from the source data files and their variables to be transferred to the variables of the metafile.

Type 1-indicators hold one single value for each source variable. For brevity, they are mostly called "indicators" in this paper. Type 2-indicators hold a list of values for each source variable. For example, the maximum value of a source variable is a (type 1-)indicator, while the levels of a source variable make a type 2-indicator.

In the resulting metafile, a type 1-indicator feeds a single variable and a type 2-indicator feeds a group of variables.

Macros and list macros

Within Stata syntax, macros serve as a kind of text module. In the course of a Stata program, strings or values are assigned to macros, in order to fulfil a certain task at different subsequent places in the syntax, for example as a loop counter or to transfer values to variables.

In Stata there are local macros that live only during the runtime of a Stata program and there are global macros that live throughout a whole Stata session. The complete syntax of SOEP-Metafile.do exclusively uses local macros.

In this paper, the term "list macro" is used for macros that collect a specific value for all variables of a source file, if it is about a type 1-indicator. In contrast, type 2-indicators fill a list macro for each variable and, consequently, a series of list macros for all variables of a source file.

Wave indicator

To report the survey year, different variables with different date formats were used in the numerous data files of the SOEP over the years. The actual SOEP standardized four-digit variable `syear <yyyy>` is not yet contained in every data file of the former waves. In addition, some of the SOEP data files hold divers different survey year variables.

For this reason, the applicable survey year variable of a source file is determined within SOEP-Metafile.do on the basis of a priority list that comprises the different wave and survey year variables that have been used in the SOEP. The wave indicator (metafile-variable `wind`)

reports the variable in the source file that has effectively been used to generate the survey year indicators in the meta file.

4 Documentation of the metafile

A metafile generated with SOEP-Metafile.do for the SOEP or SOEPlong data of version 33 (data release 2017) holds 261 variables: 33 flag variables for the survey years in which the source variables are present with non-missing values, 100 variables each for the levels and the values of the source variables, 18 variables reporting further indicators on the source variables, 4 variables holding indicators on the source data file, as well as 6 variables showing "technical" characteristics.

4.1 Synopsis of the metafile variables

- File related indicators
 - o Number of observations in the source file
 - o Number of variables in the source file
 - o Number of variables in the source file with non-missing values (SOEP-specific: with values ≥ 0 & $< .$)
 - o Wave indicator: relevant variable in the source file that holds the survey year
- Variable related indicators (type 1-indicators)
 - o Number of non-missing observations
 - o Number of system-missings
 - o Number of levels
 - o Number of non-missing levels
 - o Value of the variable if there is only 1 level
 - o Value of the variable if there is only 1 non-missing level
 - o Frequency of value 0
 - o Minimal value
 - o Minimal non-missing value
 - o Maximal value (without system missings)
 - o Name of the value label definition
 - o Number of unlabeled observations (only with labeled variables)
 - o Highest labeled value
 - o Minimal length (string variable)
 - o Maximal length (string variable)
 - o Number of survey years

- o Earliest survey year
- o Latest survey year
- Variable related type 2-indicators
 - o Survey years
 - o Levels (variables with max. 100 levels)
 - o Value labels (variables with max. 100 levels)
- "Technical" variables
 - o Consecutive number
 - o Name of the source file
 - o Name of the source variable
 - o Data type of the source variable
 - o Storage type of the source variable
 - o Variable label of the source variable

4.2 Detailed documentation of the metafile-variables

Variable	Variable label	Description
lfn	consecutive number	numVar, range: integers > 0 Consecutive number of all observations. Is generated anew at each run of SOEP-Metafile.do and can therefore not be used as a unique identifier across 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)
datenfile	source file	stringVar Name of the source data file
varname	Variable name	stringVar Name of the source data variable
type	storage type	stringVar Data type of the source variable (generated by the Stata-command describe, replace clear)
isnumeric	whether numeric or string	numVar, range: 1, 0 Flag: source variable is numeric/not numeric
varlab	variable label	stringVar Variable label of the source variable. NOTE: the name of this metafile-variable can be changed by the user in the definitions-section of SOEP-Metafile.do.

Variable	Variable label	Description
nvrs	No. of vars in file	numVar, range: integers > 0
nvrsnm	No. of vars in file with nonmissing values	numVar, range: integers >= 0
nobs	No. of obs. in file	numVar, range: integers > 0 Number of observations of the source data file
nonmis	No. of nonmiss. obs. (variable related)	numVar, range: integers >= 0 Number of observations of the source variable with values >= 0 & < . (numvars) or: empty and no missing code (stringvars) NOTE: not valid for variables that may have negative values that are no missing codes (mainly generated income and wealth variables). With string variables, empty entries or entries having a length of 2 characters and containing a minus sign, are evaluated as missing.
sysmis	No. of system-missing obs. (variable related)	numVar, range: integers >= 0 Number of observations with value system missing <dot>. Is invariably 0 with string variables.
uniqvals	No. of unique values	numVar, range: integers > 0 Number of unique values (including missing codes and sysmis <dot>) With string variables: Number of diverse entries (including "")
uniqvalpos	No. of unique values >= 0/nonmiss	numVar, range: integers > 0 Number of unique values >=0 & < . (thus without sysmis and missing codes) With string variables: Number of diverse entries which are neither empty, nor just 2 characters long whereof one is a minus sign (= SOEP-missing code)
valuniq	value if only 1 unique value	numVar, range: -900, -200, values of the source variable value of the source variable if uniqvals == 1 -900 uniqvals > 1 -200 with string variables
valuniqpos	value if only 1 unique value >= 0	numVar, range: -900, -200, values of the source variable Value of the source variable if uniqvalpos == 1 -900 uniqvalpos != 1 -200 with string variables
nvalzero	freq of value 0	numVar, range: integers >= 0 With string variables: frequency of empty strings
valmin	minimal value	numVar, range: -200, values of the source variable Decimals rounded to 2 decimal places -200 with string variables

Variable	Variable label	Description
valminp	minimal value ≥ 0	numVar, range: -900, -200, values of the source variable ≥ 0 Minimal value of the source variable that is ≥ 0 is (= minimal non-missing value). -900 valmax < 0 -200 with string variables
Valmax	maximal value	numVar, range: -200, values of the source variable Decimals rounded to 2 decimal places Without sysmis <dot>, except if sysmis is the only value (in that case, valmin is also sysmis) -200 with string variables
vallabset	name of valuelabel set	stringVar Name of the attached value label set "-2" no value label set is attached
undoc	No. of undocumented obs. (if var is labelled)	numVar, range: -9, -2, integers ≥ 0 Number of observations of the source variable in non-labeled categories, although a value label set is attached to the variable -9 the source variable has a value label set, but only missing codes and possibly value 0 are labeled. -2 the source variable has no attached value label set.
vlabmax	highest labeled value	numVar, range: -200, values of the source variable -200 if no value label set is attached NOTE: vlabmax relates to the defined value label set, not to actually observed categories of the source variable.
strmin	min. length (stringvars)	numVar, range: -2, integers ≥ 0 Minimal length of non-empty entries (= 0 if all entries in the source variable are empty) -2 with numVars
strmax	max. length (stringvars)	numVar, range: -2, integers ≥ 0 Maximal length of non-empty entries (= 0 if all entries in the source variable are empty) -2 with numVars
wind	wave indicator	stringVar Reports the wave indicator in the source file that is the basis for the generation of the metafile-variables nsurveys, syearmin, syearmax and sy1984 - sy20## Potential wave indicators in the source files are used according to the priority list: syear, welle, wave, intyear, erhebj, bioyear -1 the source file contains none of the wave indicators of the priority list

Variable	Variable label	Description
nsurveys	No. of waves	numVar, range: -1, integers ≥ 0 Number of waves in which the source variable is surveyed. Generated on the basis of the variables: syyear, welle, wave, intyear, erhebj, bioyear (rank order, the first met variable is used) 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
syyearmin	earliest survey year	numVar, range: -9, -1, integers ≥ 0 & $\leq$ survey year of the actual data release Earliest year in which the source variable was surveyed, i.e. minimal value of the relevant wave indicator, without transformation into the four-digit format yyyy -1 the source file contains none of the wave indicators of the priority list -9 nsurveys == 0 (i.e. the source file contains a wave indicator, but the source variable has no non-missing values)
syyearmax	latest survey year	numVar, range: -9, -1, integers ≥ 0 & $\leq$ survey year of the actual data release Latest year in which the source variable was surveyed, i.e. minimal value of the relevant wave indicator, without transformation into the four-digit format yyyy -1 the source file contains none of the wave indicators of the priority list -9 nsurveys == 0 (i.e. the source file contains a wave indicator, but the source variable has no non-missing values)
sy1984 - sy20##	--	numVars, range: -1, 0, 1 Flag variable for the presence of the source variable in the survey years 1 the source variable has nonmissing values in the survey year 0 the source variable has not been surveyed or has no non-missing values in the survey year -1 the source file contains none of the wave indicators of the priority list

Variable	Variable label	Description
val1 - val100	--	numVars, range: -200, -100, -900, sysmis, values of the source variable The variables val1-val100 show the values of the actually observed levels of the source variables (only variables with maximally 100 non-missing levels -100 numVar has more than100 non-missing levels -900 numVar has maximally 100 levels, but all values are < 0 (SOEP-missing Codes) -200 stringvars. (The values of the stringvars are recorded in the metafile-variables lab1-lab100, because val1-val100 are numvars and cannot hold strings) sysmis <dot> val#-variables that are not taken
lab1 - lab100	--	strVars The variable lab1-lab100 show the labels of the actually observed levels of the source variables (only variables with maximally 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 variables without attached value label set "-900" source variable has value label set and maximally 100 levels, but all values are < 0 (SOEP-missing Codes) With string variables the metafile-variables lab# hold the unique values (levels) of the source variable.

5 How to generate a metafile on the SOEP data using SOEP-Metafile.do

To generate a metafile on the SOEP data, users must adapt the do-file SOEP-Metafile.do to their local environment. What has to be done in practice, is described in detail in section 5.2 of this paper.

Before that, some insight into the functioning principles of SOEP-Metafile.do shall be provided, which will make it easier to understand the parameters that have to be set:

SOEP-Metafile.do uses all - or a selection of - the Stata data files that are stored in a directory that has to be determined by the user. These data files are processed one by one, while certain indicators for the variables in the data files are extracted. The indicators for all the variables of a specific data file are saved temporarily in a file-specific metafile. After the data files are processed, the file-specific metafiles are assembled to the overall metafile, which is the final outcome.

To minimize the runtime of the job, data files holding more than a certain number of variables are fractionised into smaller portions (sectionized processing).

In the output and appointed log files, SOEP-Metafile.do reports the advancement of the process, possible data problems preventing the processing of a file, the name and path of the resulting metafile, as well as the runtime of the job.

5.1 Hints and recommendations on the use of SOEP-Metafile.do

- The runtime of SOEP-Metafile.do in the EDV-environment of the SOEP FDZ¹ was a short hour for the wave specific SOEP files and about 3 hours for SOEPlong (data release v33, 2017), with sectionized processing and data portions of 50 variables.
- In view of the runtime, the correct functioning of SOEP-Metafile.do should be tried out in a test run. This can either be achieved by specifying a criterion in the macro files that is met only by few data files (such as: `bgp*.dta`, `*kind.dta` etc.). Or by setting up a test directory that contains only certain data files. For test runs on the big SOEPlong data files, it is highly recommended to reduce the number of observations in the files drastically, because the number of observations influences the runtime to a high degree.
- It is more efficient to generate separate metafiles for the wave-specific SOEP-files and for SOEPlong. Because, if you generate one metafile for both in one single run of SOEP-Metafile.do, the resulting file will hold records on SOEP-variables and on SOEPlong-variables, but no concise criterion to separate both if necessary is available. If you wish to have one single metafile for both data pools, you should generate them separately and

¹ FDZ: Forschungsdatenzentrum (Research Data Center)

join the two metafiles by the Stata command `append, generate(newvar)`. This procedure generates a flag-variable that signals the origin of the records as either from SOEP or from SOEPLong.

5.2 Adaptations required in the syntax segment "Definition of parameters"

The syntax of SOEP-Metafile.do is divided in several sections. Adaptations by the users are made exclusively in section "B) DEFINITION OF PARAMETERS", which follows section "A) AUTOMATIC DEFINITION OF PARAMETERS". Comments and examples that are inserted into the syntax help with the determination of the parameters.

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

In section "B) DEFINITION OF PARAMETERS" the macros are defined, which are listed in the following table. The indication "mandatory" means: the macro must not be empty but has to hold a value (i.e. the macro may not be defined as: `local xyz ""`). In contrast, optional macros may be defined as empty. The respective consequences of that are described in the table. The indication "optional, or mandatory if SYI desired" means: The macro is optional if no survey year-indicators (SYI) are desired. However, if SYI are desired, the macro must hold the appropriate value.

Most of the listed macros in the table are either defined by a default value if they have not been defined in the section "definition of parameters", or if they are already predefined in the section "definition of parameters" and either may be changed or may as well stay unchanged. Thus, the number of parameters that must be set by the user is diminished considerably, compared to the list.

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

Name of the macro	Explanation
we	Appendix to the name of the resulting metafile. Ought to be used to indicate the SOEP version that is used to generate the metafile. Optional. If the macro is defined as empty, the file name of the resulting metafile does not show the version of the SOEP data it is based on. In addition, the metafile of the previous run is overwritten unintendedly, if the actual metafile is generated in the same day and written to the same target directory (e.g. if you generate metafiles for the wave specific SOEP data and for SOEPlong). Predefined in the syntax as "_SOEPv33_long". Recommendation: Control and, if necessary, adapt the macro definition.
data	Path of the directory where the SOEP data files that are to be processed are stored. Mandatory. Note: The source files are loaded to the working memory. No changes to the original data are made, therefore there is no risk of data loss by using SOEP-Metafile.do .
pfad1 pfad1d pfad1o	pfad1d: Target directory for the resulting metafile pfad1o: Target directory for the logfile both are subdirectories to the target main directory pfad1. Mandatory. The path specifications are designed to fit the common practice to establish separate subdirectories to a main directory for data, output, and do-files (the latter not being used by SOEP-Metafile.do. All 3 macros must hold path definitions, but it is allowed to supply the same path to them (see examples in the syntax).
files	Criterion to select the files that are to be processed within the source directory. The criterion must consist of a single item with or without wildcards, i.e "bdp.dta" or "b*.dta" or "*p.dta"). Optional. If the macro is defined as empty, the default value "*.dta" is used.
port	Number of variables per portion with segmented processing. Optional. If the macro is defined as empty or has the value 0, the default value 50 is used. This value has proved to be a good trade-off between reducing the runtime by segmentized processing and augmenting the runtime by a higher number of runs over the portions. Note: If the value of port is bigger than half the number of variables in a file, the file is processed without portioning.
varlab	Name of the variable in the metafile holding the variable labels of the source variables. Optional. If the macro is defined as empty, the default-value "varlab" is used. The purpose of this macro is to hold English and German variable labels in the same metafile. If the runtime is no problem, you can generate a metafile for the English and German SOEP data versions each, and then merge the label of the other language to the master metafile using the key variables datenfile and varname.
result	Name of the resulting metafile. Predefined as: metafile`we' `datum'. Mandatory. It is allowed, but not required, to change the macro definition. Recommendation: Do not change the predefinition of the macro.
reslab	Label that is attached to the resulting metafile using the Stata-command label data. Optional. If the macro is defined as empty, no label is attached to the metafile.

Name of the macro	Explanation
lg	Name of the resulting log file. Defined as: metafile`we'`datum'.log. Mandatory. It is allowed, but not required, to change the macro definition. Recommendation: Do not change the predefinition of the macro.
syrlst	Priority list of the possible wave indicators in the source files. Predefined as: "syear welle wave intyear erhebj bioyear". Optional, or mandatory if SYI desired. It is allowed, but not required, to change the list. Changes in the rank order of the list are not reasonable. The macro may be defined as empty. This has the consequence that the resulting metafile holds no information on the appearance of the variables in the survey years. Recommendation: Do not change the predefinition of the macro.
wave1, wave2	Earliest and latest survey year, predefined as wave1 = 1984, wave2 = 2016 (= current wave of V33). Optional, or mandatory if SYI desired. The macros may stay empty if no information on the survey years is desired. If survey year indicators are desired, wave2 has to be adapted to the current wave, while wave1 stays unchanged. Note: wave1 and wave2 are also used in the syntax to discriminate valid and invalid survey years. Therefore the definition of wave1 and wave2 ought to conform with the actual earliest and latest waves in the SOEP data, if not empty. Recommendation: Don't change the predefinition of wave1. Control and, if necessary, change wave2.

6 Syntax examples on how to use the metafile

Note: In the following, i.a. string functions for data selection are presented. From Stata 14, the names of string functions have a prefixed "u" (standing for "Unicode"). The function strpos() becomes ustrpos() with Stata 14, for instance. Therefore, if you are using Stata 14 or higher, the syntax examples must be adapted accordingly.

To write syntax that works both with Stata 13 and higher Stata versions, you can define a macro that is prefixed to the function name and that contains a "u" if Stata from version 14 is the current program and is empty otherwise. You can see in the SOEP-Metafile.do-syntax in section 8 how this is implemented practically.

As far as the following examples relate to concrete data, they have been made with data from a metafile generated from SOEPlong v33 (data release 2017).

Regrettably, the examples in this documentation, the English language version, use data with German variable labels. Please accept my apologies: rerunning the examples in English would have consumed a significant amount of time.

6.1 Create views of the metafile in the Stata data editor

Using the Stata data editor, the metafile can be viewed literally as a table. Using the edit mode, you can change values manually or copy them to the clipboard, while the browse mode only allows for looking at the data. In any case, it is much more comfortable for a human viewer to restrict the displayed information to the data that is currently relevant.

6.1.1 Selection of metafile-variables

If you want to view certain variables from the metafile, it is often advisable to define narrower display formats before editing or browsing, so that more columns fit into the screen area:

```
format lfn %6.0g
format datenfile varname %8s
format varlab %46s
format nonmis uniqvalpos valminp valmax nsurveys syearmin syearmax %6.0g
format sy1984-sy2016 %7.0g
edit lfn datenfile varname varlab nonmis uniqvalpos valminp valmax nsurveys
syearmin syearmax sy1984 sy1985 sy1986
```

6.1.2 Selection of observations

```
edit <varlist> if <conditions>
```

or:

```
browse <varlist> if <conditions>
```

The Stata expressions to select string variables are less known than the ones to select numeric variables. As the metafile contains quite a few string variables, the most important string functions for data selection are presented by the following examples.

- ... if the string "erwerbs" is contained in varlab:

```
...if strpos(varlab, "erwerbs") > 0
```

The function `strpos()` returns the value 0, if the sought after string is not contained within varlab.

- ... if the string "erwerbs" or "Erwerbs" is contained in varlab:

```
...if strpos(strlower(varlab), "erwerbs") > 0
```

In this example the string functions `strpos()` and `strlower()` are nested. The expression signifies: ... if the string "(...)" is contained in varlab after the transformation in lower case characters (equivalent to: no matter if it is put in upper or lower cases).

- ...if the variable contains a string that begins with "bgp":

```
...if strpos(datenfile, "bgp") == 1
```

or:

```
...if substr(datenfile, 1,3) == "bgp"
```

The first version asks if the string "bgp" begins at the first position of the content of variable datenfile. The second version asks if three characters of the variable content, beginning with the first character, are made out of "bgp".

See more details and further string functions using the Stata command help string functions.

6.2 Selection of variables from the SOEP Long-files

For a still not precisely outlined research project, the SOEPlong data can - as an example - be explored for variables that have been surveyed in at least 5 waves since 2008 and that have at least 500 non-missing observations. The successive examples show a (simplified) process of concretisation of the final research issue.

Attention: Potential pitfall! You might be tempted to put the condition as follows:

```
... if syearmin >= 2008 & nsurveys >= 5
```

This would be wrong, because the condition excludes those variables that comply with the above named requirements, but were also surveyed in waves before 2008.

Instead, we generate an auxiliary variable that indicates how often a variable has been surveyed since 2008:

```
capture drop h1
egen h1 = anycount(sy2008-sy2016), values(1)
```

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

```
ed lfn datenfile varname varlab nonmis nsurveys syearmin syearmax if h1 > 4 &
nonmis >= 500
```

→ This condition is met by 1,949 variables in SOEP long (V33).

A visual inspection shows that the selected set contains variables relating to "satisfaction". These shall be examined closer (note: "Zufriedenheit" is the German word for "satisfaction"):

```
ed lfn datenfile varname varlab nonmis h1 if strpos(varlab, "ufriedenh") > 0 &
nonmis >= 500
```

→ the selected set contains 5 variables from the file jugendl and 12 variables from the file pl. We decide on using only data from pl, but we need to investigate if there are further satisfaction-related variables in pl that were excluded by our selection criterion. In any case, we want to hold on to the criterion "at least 500 non-missing observations". However, we lower the number of waves in which the variables have been surveyed within the relevant period to at least 1:

```
ed lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1 if datenfile
== "pl" & nonmis >= 500 & h1 > 0 & strpos(varlab, "ufriedenh") > 0
```

To define macros for the repeatedly used components is a more elegant way to deal with long Stata commands:

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond "datenfile == \"pl\" & nonmis >= 500 & h1 > 0"

ed `vars' if `cond' & strpos(varlab, "ufriedenh") > 0
```

→ This condition results in a set of 37 variables. We sort these according to how often they were surveyed during the relevant period. Our second sort criterion is varname, because variables belonging to the same item set usually have consecutive variable names in the SOEP data:

```
sort h1 varname
```

After inspection in the data editor, we decide to use the previously selected satisfaction-variables, which have been surveyed 9 times in the relevant period, plus the 4 variables that were only surveyed in 2016 and that measure the "satisfaction with [...] in the last 10 years (German variable label: "Zufriedenheit mit [...] in den letzten 10 Jahren"). We mark these variables with a flag variable (see next section).

6.3 The use of an auxiliary variable to mark selected variable sets

It may be difficult or even impossible to retrieve a certain variable set using a concise selection criterion. For example, the applied selection criterion may retrieve - besides the desired hits - some variables that do not belong to the targeted set. Or otherwise, the selection criterion is already very complex and you need to add further conditions. In such cases you can generate a flag variable that marks if a variable belongs to the targeted set. The flag can be set or unset stepwise.

While it is possible to change the flag variable manually in the data editor, it is strongly recommended not to do so. Instead, executing every step using the Stata syntax editor and saving the steps to a sps-file, makes your data preparation reproducible.

To continue the above example:

First, we test with edit if the selection criterion is correct and leads to the targeted variable set:

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond1 "datenfile==\"pl\" & nonmis >= 500 & strpos(varlab, \"ufriedenh\") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, \"letzte 10\") > 0)"

ed `vars' if `cond1' & `cond2'
```

After that we use it to generate the flag variable:

```
capture drop f1
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
gen f1 = cond(`cond1' & `cond2', 1, 0)
```

alternatively, if you are not familiar with the cond() syntax:

```
capture drop f1
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
gen f1 = 0
replace f1 = 1 if `cond1' & `cond2'
```

Recommendation: control the result always with edit if f1==1

We notice an odd gap between lfn 7520 (Variable plh0180) and lfn 7522 (Variable plh0182). Why did our selection criterion not retrieve the variable in between with lfn 7521? (Note: if you have not dropped any observations, the variable lfn contains a consecutive number without gaps).

```
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
local cond1 "datenfile=="pl" & nonmis >= 500 & strpos(varlab, "ufriedenh") > 0"
local cond2 "h1==9 | (h1==1 & syearmax==2016 & strpos(varlab, "letzte 10") > 0)"
ed `vars' if (`cond1' & `cond2') | lfn == 7521
```

→ We see the reason why the variable was excluded: it was surveyed in the relevant period only 4 times, instead of 9 times like the adjacent variables. One of our condition was: h1 == 9 or (h1 == 1 and (...)). If we decide on including this variable in the target set, we mark it in the flag variable f1:

```
replace f1 = 1 if lfn == 7521
```

As a rule you should examine thoroughly if the result is in accord with the desired target set. This is most important if the variable label is part of the selection criterion and if you do not explore SOEPlong variables but wave specific variables. Typing errors and other irregularities may easily prevent a variable that belongs to the target set from being included in the selected set.

It is also highly recommended to have a look at the variables that neighbour the finally selected variables in order to identify and find those variables that do not meet the selection criteria but belong semantically to the selected set (as was the case in the above example). For instance, rarely stated items from item sets or variables that mark the overall-non-response of an item set may easily fall through the selection grid. The easiest way to find neighboured variables is by adjacent values of lfn.

The variable selection to date has the running numbers (lfn) 7511 through 7522 and 7676 through 7679. We use these to view the adjacent variables:

```
sort lfn
local vars "lfn datenfile varname varlab nonmis nsurveys syearmin syearmax h1"
edit `vars' if (lfn > 7505 & lfn < 7528) | (lfn > 7670 & lfn < 7685)
```

→ We can see that there are no variables that belong directly to our set of selected variables. However, we see variables on the frequency of certain emotions in the last 4 weeks. We add these to our selection set ("letzte 4" is "last 4" in German):

```
local cond3 "(lfn > 7505 & lfn < 7528) | (lfn > 7670 & lfn < 7685)"
replace f1 = 1 if (`cond3') & strpos(varlab, "letzte 4") > 0
```

Please note the additional brackets that enclose `cond3' in the replace-command. Without these, Stata would interpret the condition as follows: Either lfn lies between 7505 and 7528 OR lfn lies between 7670 and 7685, and the string "letzte 4" is found in varlab. This is not what we want. We want the instruction "letzte 4" is found in varlab to be applied to both number ranges.

This is another instance of how important it is to carefully verify that the applied commands return the desired results.

Thus, after the desired variable set is defined, in the next section you will see how a list of the selected variables can be created for use in self written syntax.

6.4 How to capture the selected variable names for further uses

To capture the names of the selected variables, we use the Stata command `levelsof`, which returns the levels of variables, since the names of the SOEP-variables are levels of the metafile-variable varname:

```
quietly levelsof varname if f1 == 1, clean local(vars)
global myvars "`vars'"
display "${myvars}"
```

The first line of this command sequence assigns the variable names to the local macro vars. The option `clean` causes the suppression of the quotation marks that would otherwise surround the variable names in the list.

The command `levelsof` can assign levels of variables to local macros only. In order to keep the variable list available during the complete Stata session, it is assigned to the global macro `myvars` in the second syntax line.

To work with the variables of this list, further variables, like IDs and weighting factors, are needed. Absolutely necessary are the key variables that serve as unique identifiers for a case in the source files - with `SOEPLong`, these are `pid` and `syar`. Either the additional variables can be marked in the flag variable, as shown above, or you add it to the list, as shown in the syntax below.

```
quietly levelsof varname if f1 == 1, clean local(vars)
global plmyvars "hid cid pid syar `vars'"
```

Note: the weighting factors (in German: Hochrechnungsfaktoren, Abbreviation hrf) can be retrieved in the wave specific files as well as in SOEPlong by the criterion if strpos(varname, "hrf") > 0. In SOEPlong, these are located in the data files hpfad und ppfad. Therefore, they cannot be included into the above compiled list, which is used to select variables from a single SOEP data file. How to proceed if the needed variables are located in different data files, is described in section 6.5.

This is how you use the variable list to retrieve the selected variables from the SOEPlong file pl:

```
use ${plmyvars} using pl, clear
```

6.5 Proceeding if the selected variables are located in several data files

The easiest way to do this is to define a separate list for each data file, either as described above or manually. It is recommended to flag the related data file and possibly the content in the name of the list, in order to not confuse the lists:

```
global hrfppfadl "phrf phrfe"
```

This list of the weighting factors and the above defined list plmyvars are then used to merge the needed variables into a target data file:

```
use ${plmyvars} using pl, clear
merge 1:1 pid syear using ppfadl, keepusing ${hrfppfadl}
```

If the needed variables are located in a multitude of data files, as may often be the case with wave specific files, you would generate a flag variable in the metafile and mark all the needed variables, no matter in which data file they are located, in exactly the same way as described above.

After that you generate a list of the related data files with the command levelsof datenfile... In a second step this list is used to generate the file-specific variable lists (for example). For this, the procedure loops over the entries in the list of data files. The counter of the loop is obtained from the number of entries in this list. The metafile is still the active data set, f1 is the flag variable for the needed variables:

```
quietly levelsof datenfiles if f1 == 1, clean local(files)
global myfiles "`files'"
local nf : word count `files'
forvalues i = 1(1)`nf' {
    local f : word `i' of `files'
    quietly levelsof varname if f1==1 & datenfile == "`f'", clean local(vars)
    global `f'vars "`vars'"
}
```

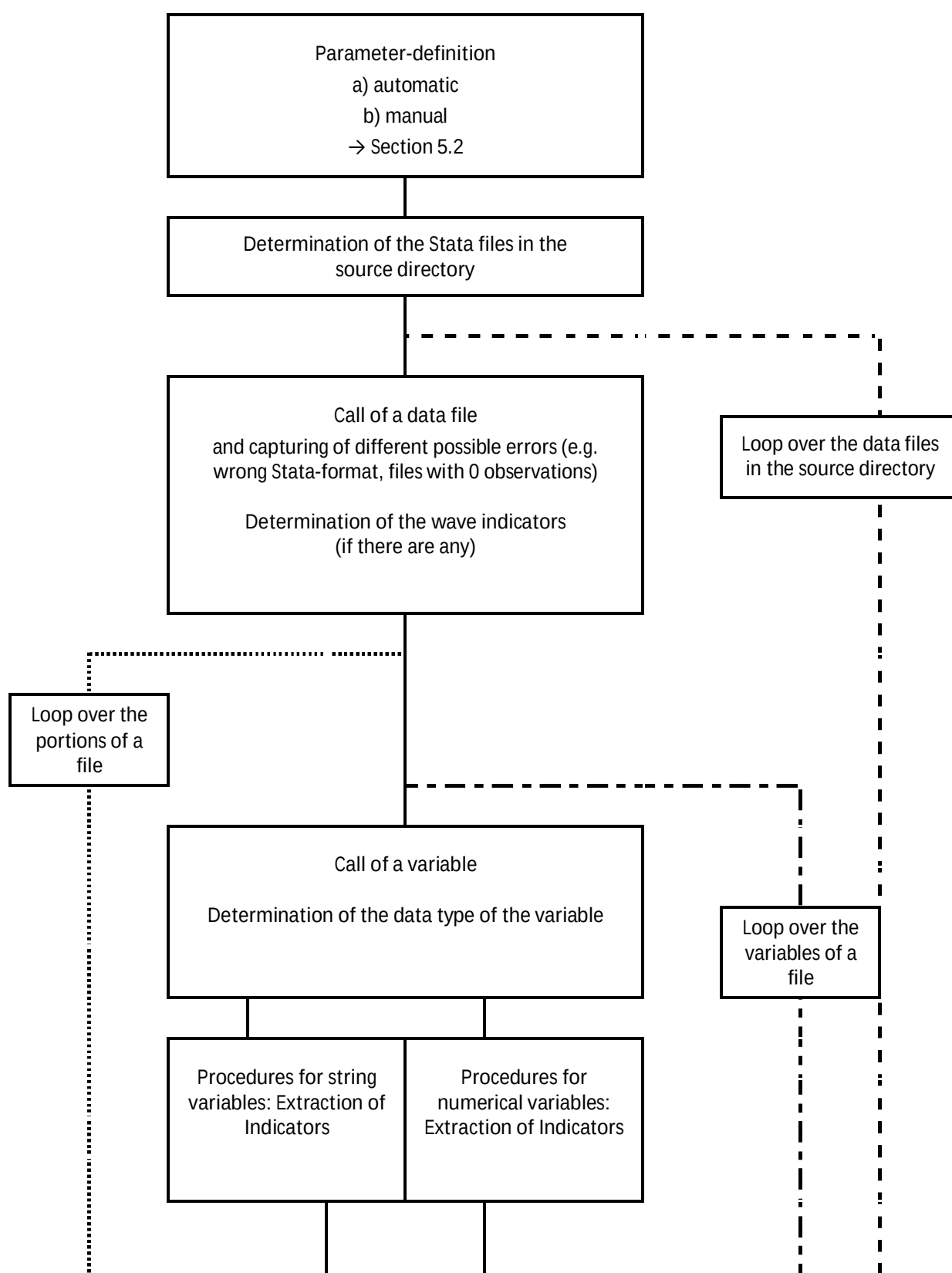
In this syntax sequence, the variable lists in local macros, which have been generated with levelsof varname, are copied to global macros so that they are still available after the

execution of this commands. In the same step, the variable lists are renamed to a data file-specific name.

To continue, the metafile is no longer needed because all relevant information has been transferred to macros. Therefore, the file specific variable lists can be used to compose the commands to call the related data files and do whatever you want to do for your analyses. However, note that the data file list contains the pure file names, without paths, which have still to be added in the commands.

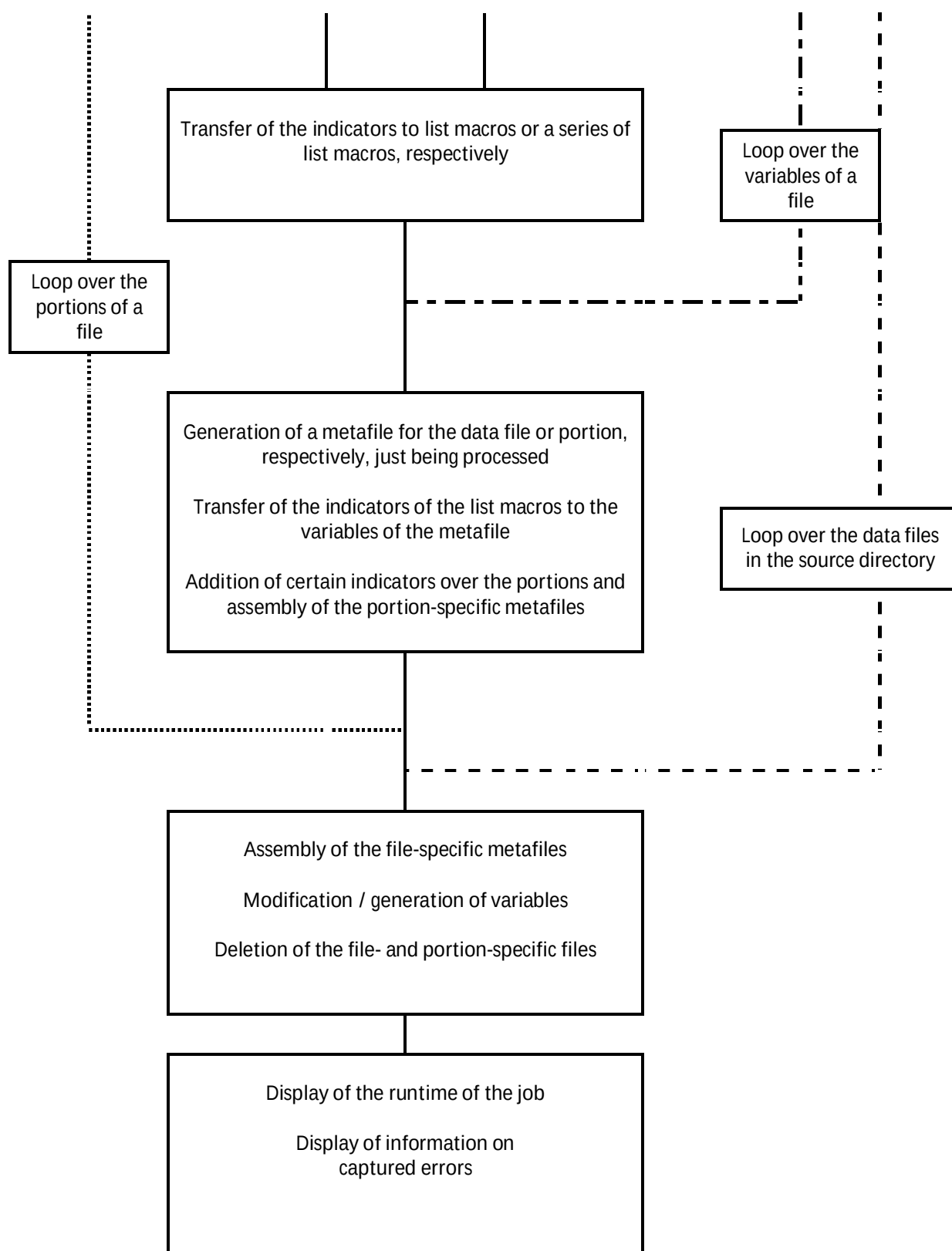
The syntax examples of this chapter are intended to guide further uses for variable and file lists that are generated from the metafile. For example, you could write syntax to automatically generate and save do-files. However, the paper at hand does not go deeper into this procedures.

7 Flow chart of SOEP-Metafile.do



(continue next page)

(continue of previous page)



8 Complete syntax of SOEP-Metafile.do

```

/* SOEP-Metafile.do, v3.0.3 Klaudia Erhardt, last updated: 2018-05-26

Version for SOEP-Users

#####

Syntax compiles a Metafile from SOEP data files in a directory.

It is highly advisable to create two separate metafiles for SOEP- and SOEPlong-files

Approximative runtime of this job:
- 1 hrs when applied to all SOEP wave-specific files
- 3 hrs when applied to all SOEPlong files (portion size: 50). Larger portion sizes mean longer runtime

*/

/*#####
#####
##### A) AUTOMATIC DEFINITION OF PARAMETERS - #####
##### no changes please!!! #####*/

clear
set more off
set varabbrev off, permanently
set output error

timer clear
timer on 1      /* Measuring the runtime of this job */
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) DEFINITION OF PARAMETERS
#####
##### */

/* ##### affix for file names ##### */

local we "_SOEPv33_long" /* Label for wave and version. Will be appended to the resulting files' names. Must
                           comply with the file name conventions of Stata, i.e. no period character.
                           May be defined as empty, but then you have to take care to prevent unintentional
                           overwriting of resulting metafiles. */

/* ##### Paths and directories #####

NOTE: specification of paths without ending slash ! */

/* Path to source-files directory - mandatory */

local data ""

/* Examples:
local data "//hume/soep-data/DATA/soep33/stata"
local data "c:\Users\myname\soep\soeplong"
*/

/* Paths to resulting files directories - all three are mandatory, but may contain the same path */

local pfad1 "" /* Main target path */
local pfad1d "`pfad1'/Datenfiles" /* local data directory for resulting metafile */
local pfad1o "`pfad1'/Output" /* local output directory - storage location for logfile */

/* Examples:

A) all resulting files are written to the data directory:
local pfad1 "`data'"
local pfad1d "`data'"
local pfad1o "`data'"

```

```

B) resulting files are written to different subdirectories of main directory:
local pfad1 "//smith/users/kerhardt/Zuwanderer/Datenverknuepfung_STATA/SOEP_Variablenuebersicht" /* Main loc path */
local pfad1d "`pfad1'/Datenfiles" /* local data directory for resulting file */
local pfad1o "`pfad1'/Output" /* local output directory - storage location for logfile */

/* ##### Source files ##### */

local files "*.dta" /* selection criteria within the files of the source directory (`data')
                    you may use wildcards to include only a subset of files.
                    If defined as empty, default "*.dta" is put into effect. */

/* ##### Options ##### */

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.
              If "port" > than half the number of variables in a data file, the processing of this
              file will be non-segmented. */

local varlab "varlab_de" /* Name of the variable in the metafile that contains the variable labels of the data
                        files - if defined as empty, default "varlab" is put into effect. */

/* ##### Target file names ##### */

local result "metafile\we'_datum'" /* resulting data set from this syntax */
local reslab "" /* Label for the resulting data set, may stay empty */
local lg "metafile\we'_datum'.log" /* log-file */

/* ##### Specification of wave or survey-year indicators ##### */

/* Priority list of possible wave indicators: highest priority first. */
local syrlist "syear wellle wave intyear erhebj bioyear" /* Usually you don't have to change this, but may stay empty
                                                         or can be uncommented if no survey year indicators are wanted. */

local wave1 "1984" /* earliest wave - may stay empty if no survey year indicators are wanted. */
local wave2 "2016" /* newest wave - may stay empty if no survey year indicators are wanted. */

```

```

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

capture log close
log using "`pfad10'/'lg'", replace

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

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

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

if `u'strlen("`files'") == 0 {
    local files "*.dta"
}

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 to come after the defi-
                                nition of wave1 and wave2! Otherwise a variable sy0 will be generated if no survey
                                year variables are wanted */

if `u'strlen("`wave1'") == 0 {
    local wave1 "0"
}
if `u'strlen("`wave2'") == 0 {
    local wave2 "0"
}
if `u'strlen("`varlab'") == 0 {
    local varlab "varlab"
}

```

```

/* ##### */

local message "      " _n ///
"The data files to be included are determined. " _n ///
"After that, a metafile is generated from each data file. Subsequently, the single meta files " _n ///
"are compiled to a joint metafile which contains data to all variables of all included dtafiles. " _n ///
"      " _n ///
"Processing... Please wait. " _n ///
"      "

set output proc
display "`message'"
set output error

local infile : dir "`data'" files ``files'
local a : word count `infile'
local infile = `u'substr("`infile'",`"',`"',`a'*2) /* remove quote signs */
local infile = `u'substr("`infile'",`.dta`,`"',`a') /* remove file extension .dta */

local message "      " _n ///
" All data files in directory `data' will be included " _n ///
"      that comply with criteria: `files' "

set output proc
display " `message' "
set output error

local nds : word count `infile'
local a = `nds'
local message " `a' data files will be processed"
set output proc
display " `message' "
set output error

local svd "" /* updated list, in case a data file from the infile-list is not found or cannot be opened */
local nvrs = 0 /* local to contain the no of vars of a file (for segmented processing) */
local nvrsnm = 0 /* local to contain the no of nonmissing vars of a file (for segmented processing) */

forvalues i = 1(1)`a' { /* Loop over the files */
    local datei = word("`infile'",`i')
    capture quietly describe using "`data'/'`datei'.dta", varlist

    if _rc == 601 {
        local fnexist "`fnexist' `datei'" /* list: files not found */
    }
}

```



```

if _rc == 610 {
    local s14file "`s14file' `datei'" /* list: files could not be opened */
}
if _rc == 0 { /* if file exists */

    local allvars = r(varlist)
    local nsik = r(k)
    local n = `nsik'
    local message "Processing file `i', file name: `datei'.dta containing `n' Vars at $S_TIME"
    set output proc
    display "`message'"
    set output error

    /* Identification of the wave indicators that are present in the actual date file. Here: without
       consideration of the priority. Only for loading the wave indicators in each portion of the
       segmented file.
    */
    if `u'strlen("`syrlist'") > 0{
        local syrl ""
        local ns : word count `syrlist' /* above defined list of wave indicators */

        /* Generation of the list of wave indicators that in fact are present in the file at hand */
        forvalues j = 1(1)`ns' { /* loop over the wave indicators */
            local syr = word("`syrlist'", `j')
            forvalues l = 1(1)`n' { /* loop over the variables of the file at hand */
                local av = word("`allvars'", `l')
                if "`av'" == "`syr'" {
                    local syrl "`syrl' `syr'"
                }
            }
        }
        local nsyrl : word count `syrl' /* list of the wave indicators that in fact are present */
    }
    /* Segmentation of big files */
    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 the portions */
    local nvrs = 0 /* local macro for the nVars of a file with segmetized processing */
    local nvrsnm = 0 /* local macro for the nonmissing nVars of a file with segmetized processing */

    local p = word(`"'list'", `s')
    if `z' == 1 {
        capture use `p' using "`data'/`datei'.dta", clear
    }
    if `z' > 1 {
        capture use `p' `syrl' using "`data'/`datei'.dta", clear
    }
    if r(N) == 0 { /* If the file has no observations */
        if `u'strpos("`nobsfile'", "`datei' ") == 0 {
            local nobsfile "`nobsfile' `datei' " /* List of files with no observations */
        }
    }
    if r(N) > 0 { /* If the file has observations */
        if `s' == `z' { /* in the last portion */
            local svd = `u'strpos("`svd' `datei'")
        }
        if `z' > 1 {
            local message "      Processing `datei'.dta, portion `s' of `z': `p' at $$_TIME"
            set output proc
            display "`message'"
        }
    }
}

```

```

        set output error
    }
    /* compile variable list, determine no of entries in the list, determine loop counter, */
    quietly describe, varlist
    local allvars = r(varlist)
    local nvars : word count `allvars' /* you could also use r(k) instead */
    local n = `nvars'

/* ##### Compile lists of the information that is to be transmitted to the metafile:#####
##### Indicators related to variables are compiled in macro lists (1 entry per #####
##### variable) while looping over the variables of the file #####
##### Indicators related to the whole file are determined outside the loop over #####
##### the variables. ##### */

/* the following lists have to be emptied before a new 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 ""

/* No of vars with nonmissing values. Will be incremented when looping over the variables, therefore
   set to 0 before new file ist processed */
local nv = 0

/* Determine the wave indicator with the highest priority in the file at hand. The procedure runs
   through the priority list from last to first entry. Every time a wave indicator is found to have

```

```

        nonmissing values in the file at hand, syv is updated. Therefore at the end syv contains the
        name of the wave indicator with the highest priority.
*/

if `u'strlen("`syrlst'") > 0 {
    local syv ""
    local ns : word count `syrlst' /* the priority list of the wave indicators */

    forvalues j = `ns'(-1)1 {
        local syr = word("`syrlst'", `j')
        capture confirm variable `syr', exact
        if _rc == 0 { /* `syr' is present in the file at hand */
            quietly summarize `syr', meanonly
            local min = r(min)
            local max = r(max)
            if r(min) < . & r(max) > 0 { /* `syr' has values between 0 and . */
                local syv "`syr'"
            }
        }
    }
}

/* ##### Start of variable-related routines: extraction of indicators and #####
   ##### transfer to macro lists ##### */

forvalues k = 1(1)`n' {
    /* Empty all var-related "messenger" macros or count entries in lists */
    local vl "" /* name of attached value label set */
    local vlmx "" /* highest labeled value */
    local uv "" /* no. of unique values */
    local uvp "" /* no. of unique values >= 0 */
    local ln "" /* minimal length of stringvar */
    local lx "" /* maximal length of stringvar */
    local sms "" /* no. of system-missings ( >= . ) */
    local nms "" /* no. of non-missing obs. */
    local sys "" /* no. of surveys (waves) where `var' has nonmissing values */
    local syn "" /* earliest survey year */
    local syx "" /* latest survey year */
    local vu "" /* value, if only 1 unique value */
    local vup "" /* value, if only 1 unique value >= 0 */
    local nz "" /* frequency of value 0 */
    local ud "" /* undocumented value (if value labels are attached to variable) */

```

```

local vx ""      /* maximum */
local vn ""      /* minimum */
local vp ""      /* minimal value >=0 */
local sylev ""   /* survey years, in which the variable has nonmissing values */

/* process first (next) variable of the list of all variables */
local var = word("`allvars'", `k')
/* determine data type of `var' */
local dt : type `var'

/* name of attached value label set and highest labeled value */
local vlx : value label `var'

if "`vlx'" == "" { /* if no label set is attached to `var' */
    local vl "-2"
    local vlmx "-200"
}
if "`vlx'" != "" { /* if a label set is attached to var... */
    quietly capture label list `vlx'

    if _rc > 0 { /* ... but the label definition does not exist */
        local vl "-2"
        local vlmx "-200"
    }
    if _rc == 0 { /* ... and the label definition exists */
        local vl "`vlx'"
        local vlmx = r(max)
    }
}

/* Number of unique values of `var' */
sort `var'
capture drop h1
by `var': gen byte h1 = cond(_n==1, 1, 0)
quietly summarize h1, meanonly
local uv = r(sum)

/* ##### data-type depending indicators: routines for stringvars ##### */
if `u'strpos("`dt'", "str") > 0 {

    /* max und 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
}

/* no. of non-missings (neither empty nor missing-code) and of value 0 (= empty strings) */
capture drop h2
gen h2 = cond(`u'strlen(`u'strtrim(`var')) == 2 & `u'strpos(`var', "-") > 0, -1, 1)
quietly replace h2 = 0 if `u'strtrim(`var') == ""
quietly inspect h2
local nms = r(N_pos)
local nz = r(N_0)

/* no. of values >= 0 (file is still sorted by `var' )
   uses the previously generated variable h2
   (h2 == 1 if stringvar is neither empty nor missing code)
   The "throwaway-variable" h3 indicates obs. with the first new value
*/
capture drop h3
by `var': gen byte h3 = cond(h2 == 1 & _n==1, 1, 0)
quietly replace h3 = 0 if `var' == "" /* empty values are not to be counted */
quietly summarize h3, meanonly
local uvp = r(sum)

/* Survey year indicators - uses the previously generated variable h2 */
if `u'strlen("`syv'") > 0 { /* wave indicator is found in file */
    quietly inspect `syv' if `syv' > 0 & `syv' < . & h2 == 1 /* if wave indicator and var
                                                                are both nonmissing */
    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 { /* no wave indicator is found in file */
    local sys = -1
    local syn = -1
    local syx = -1
}

/* List for the 100 value label variables */
quietly levelsof `var' if h2 == 1 & `uvp' < 101 , clean separate(*) local(labels)
if `u'strlen("`labels'") > 0 {
    local labels = "`labels'"
}

/* assign ersatz values for indicators that do not apply to stringvars */

local vu = -200
local vp = -200
local vup = -200
local ud = -2
local vx = -200
local vn = -200
local vlmx = -200
local sms = 0

} /* end: routines for stringvars */

/* ##### data-type depending indicators: routines for numvars (= no stringvars) ##### */

/* default missings for max-min-str-length, unique value (file is still sorted by var) */
if `u'strpos("`dt'", "str") == 0 {
    local ln = -2
    local lx = -2
    if `uv' == 1 {
        local vu = `var'[1]
    }
    if `uv' != 1 {
        local vu = -900
    }
}

/* no. of unique vlaues >= 0 (file is still sorted by var) */
capture drop h2

```

```

by `var': gen byte h2 = cond(`var' >= 0 & `var' < . & _n==1, 1, 0)
quietly summarize h2, meanonly
local uvp = r(sum)

/* positive value, if `var' has only 1 positive value */
if `uvp' == 1 {
    gsort -h2 /* ATTENTION: re-sorting of file */
    local vup = `var'[1]
}
if `uvp' != 1 {
    local vup = -900
}

/* Freq. of value 0, of nonmiss and of unlabeled values */
quietly inspect `var'
local nz = r(N_0)
local ud = r(N_undoc)
if `ud' == . {
    local ud = 0
}
local nms = r(N_0) + r(N_pos)

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

/* maximal and minimal value of `var' */
quietly summarize `var', meanonly
local vx = round(r(max), .001) /* rounded in order to not exceed the max length of macros */
local vn = round(r(min), .001) /* rounded in order to not exceed the max length of macros */

/* minimal positive value of `var' */
quietly summarize `var' if `var' >= 0 & `var' < ., meanonly
local vp = round(r(min), .001) /* rounded in order to not exceed the max length of macros */
if `vp' == . { /* if `var' has no values >= 0 */
    local vp = -900
}

```



```

/* ##### Survey year indicators #####
no. of survey years and earliest and latest survey year */

if `u'strlen("`syv'") > 0 { /* wave indicator is found in file */
    quietly inspect `syv' if `syv' > 0 & `syv' < . & `var' >= 0 & `var' < .
    local sys = r(N_unique)
    quietly summarize `syv' if `syv' > 0 & `syv' < . & `var' >= 0 & `var' < ., 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 indicator is found in file */
    local sys -1
    local syn -1
    local syx -1
}

/* ##### Generation of the sy####-, val####-, and lab### vars #####
##### fill macro lists for each of them ##### */

/* ##### Generation of the survey year lists ##### */

/* create list of survey years where `var' has nonmissing values (not yet standardized).
the option missing of levelsof effects that level "." von syv is included */

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

/* ##### Generation of the lists for values and value labels ##### */

quietly levelsof `var' if `var' >= 0 & `u'vp' < 101, local(values)
local c : word count `values'
if `c' > 0 { /* if `var' has nonmissing unique values from 1 to 100 */
    if "`v'lx'" != "-2" { /* if a labelset is attached to `var' */
        local labels ""
        forvalues j=1(1)`c' {
            local wx : word `j' of `values'
            local lab : label (`var') `wx', strict
        }
    }
}

```

```

        local lab ``lab'***' /* asterix used as a delimiter for the entries because
                                labels in SOEP do not contain any asterixes */
        if ``lab' == "" {
            local lab "###*"
        }
        local labels ``labels'`lab'
    }
}

} /* end: routines for numvars */

/* Compile lists: each has a value for each variable */
local uval ``uval' `uv' /* no. of unique values */
local uvalp ``uvalp' `uvp' /* no. of unique values >= 0 */
local lmin ``lmin' `ln' /* minimal length of stringvar */
local lmax ``lmax' `lx' /* maximal length of stringvar */
local valu ``valu' `vu' /* value, if only 1 unique value */
local valup ``valup' `vup' /* value, if only 1 unique value >= 0 */
local nzero ``nzero' `nz' /* frequency of value 0 */
local vmax ``vmax' `vx' /* maximum */
local vmin ``vmin' `vn' /* minimum */
local vminp ``vminp' `vp' /* minimal value >=0 */
local nmiss ``nmiss' `nms' /* no. of non-missing obs. */
local smiss ``smiss' `sms' /* no. of system-missings */
local nsvys ``nsvys' `sys' /* no. of surveys (waves) where `var' has nonmissing values */
local symin ``symin' `syn' /* earliest survey year */
local symax ``symax' `syx' /* latest survey year */
local undc ``undc' `ud' /* undocumented value (if value labels are attached to variable) */
local vlab ``vlab' `vl' /* name of attached value label set */
local vlabmx ``vlabmx' `vlmx' /* highest labeled value */

/* compile lists: each list has several values for one variable. The lists are distinguished by
the index k of the k-th variable */

/* compile list of standardised survey years for k-th variable */
local sytemp "" /* list of standardised survey years for k-th variable */
if ``sylev' != "" { /* list of survey years where `var' has values >= 0, filled above */
    local c : word count `sylev'
    forvalues j = 1(1)`c' {

```

```

    local sla : word `j' of `sylev'
    local slan = real("`sla'")

    /* treatment of disparate formats of wave indicators in data */
    if `slan' > 0 & `slan' < 84 {
        if `u'strpos("`datei'", "ost") > 0 {
            local slan = `slan' + 6
            local slan = `slan' + 1983
        }
        if `u'strpos("`datei'", "ost") == 0 {
            local slan = `slan' + 1983
        }
    }
    if `slan' >= 84 & `slan' <= 99 {
        local slan = `slan' + 1900
    }
    /* more inadmissible values of `slan' are filtered out and compiled in a list
        further down */

    local sytemp "`sytemp' `slan'" /* compile the list of standardised survey years */
}
local sylev`k' "`sytemp'"
}

/* k-th list of values if `var' has max. 100 nonmis unique values */
local c : word count `values'
if `c' > 0 {
    local values`k' "`values'"
}

/* k-th list of labels if `var' has max. 100 nonmis unique values */
if `u'strlen("`labels'") > 0 {
    local labels`k' "`labels'"
}

/* update counting for no. of nonmissing vars */

if `nms' > 0 {
    local nvn = `nv' + 1
}
}

/* ##### End: routines for the variables of a file, Start: routines for a file (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. obs */
local nv = c(k) /* No. of vars */
/* (no. of vars in file with nonmissing values is updated before the end of the variable loop) */

/* Generate new file where each variable of the data file is an observation */
describe, replace clear
if c(N) > 0 { /* if there are variables in the file ... */
    keep name varlab type isnumeric
    rename name varname
    rename varlab `varlab'
    /* Indicator of the data file the variable springs from */
    capture drop datenfile
    gen str datenfile = "`datei'"
    lab var datenfile "source file"
    order datenfile, first
    /* Number of variables (file related) */
    capture drop nvrs
    gen nvrs = `nv'
    lab var nvrs "No. of vars in file"
    /* Number of variables with nonmissing values (file related)*/
    capture drop nvrsnm
    gen nvrsnm = `nvn'
    lab var nvrsnm "No. of vars in file with nonmissing values"
    /* Number of observations (file related)*/
    capture drop nobs
    gen nobs = `no'
    lab var nobs "No. of obs. in file"
    /* Number of nonmissing observations */
    capture drop nonmis
    gen long nonmis = 0
    lab var nonmis "No. of nonmiss. obs. (variable related)"
    /* Number of Sysmis-Observationen */
    capture drop sysmis
    gen long sysmis = 0
    lab var sysmis "No. of system-missing obs. (variable related)"
    /* Number of unique values */
    capture drop uniqvals

```

```

gen uniqvals = 0
lab var uniqvals "No. of unique values"
/* Number of unique values >= 0 bzw. nonmissing */
capture drop uniqvalpos
gen uniqvalpos = 0
lab var uniqvalpos "No. of unique values >= 0/nonmiss"
/* Value if only 1 unique value */
capture drop valuniq
gen long valuniq = .
lab var valuniq "value if only 1 unique value"
/* positive value if there is only 1 positive unique value */
capture drop valuniqpos
gen long valuniqpos = .
lab var valuniqpos "value if only 1 unique value >= 0"
/* frequency of value 0 (numvars) */
capture drop nvalzero
gen long nvalzero = .
lab var nvalzero "freq of value 0"
/* minimal value of variable */
capture drop valmin
gen long valmin = .
lab var valmin "minimal value"
/* minimal value of variable >= 0 */
capture drop valminp
gen long valminp = .
lab var valminp "minimal value >= 0"
/* maximal value of variable */
capture drop valmax
gen long valmax = .
lab var valmax "maximal value"
/* name of attached valuelabel set */
capture drop vallabset
gen str vallabset = ""
lab var vallabset "name of valuelabel set "
/* undocumented Observations (if value labels are attached to variable) */
capture drop undoc
gen long undoc = .
lab var undoc "No. of undocumented obs. (if var is labelled)"
/* highest labeled value */
capture drop vlabmax
gen vlabmax = .
lab var vlabmax "highest labeled value"
/* minimal length (string variables) */
capture drop strmin

```

```

gen long strmin = .
lab var strmin "min. length (stringvars)"
/* maximal length (string variables) */
capture drop strmax
gen long strmax = .
lab var strmax "max. length (stringvars)"

/* variables for 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 nsurveys
gen byte nsurveys = 0
lab var nsurveys "No. of waves"
capture drop syearmin
gen syearmin = 0
lab var syearmin "earliest survey year"
capture drop syearmax
gen syearmax = 0
lab var syearmax "latest survey year"

/* generate blank variables for the survey years */
local n = `nw'
forvalues j = 1(1)`n' {
    local scrpt = `j'-1+'wave1'
    capture drop sy`scrpt'
    gen int sy`scrpt' = 0
}
/* generate blank variables for the values (only vars with 1-100 unique values) */
forvalues j = 1(1)100 {
    local scrpt = `j'
    capture drop val`scrpt'
    gen val`scrpt' = .
}
/* generate blank variables for the value labels */
forvalues j = 1(1)100 {
    local scrpt = `j'
    capture drop lab`scrpt'
    gen str lab`scrpt' = ""
}

```

```
/* Note: the variables undoc, val# and lab# are modified further down, after the single
files have been assembled */
```

```
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 `smis'
    local sys : word `k' of `nsvys'
    local syn : word `k' of `symin'
    local syx : word `k' of `syx'
    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 nsurveys = `sys' if varname == "`var'"
        replace syearmin = `syn' if varname == "`var'"
        replace syearmax = `syx' if varname == "`var'"
        replace undoc = `ud' if varname == "`var'"
        replace vallabset = "`vl'" if varname == "`var'"
        replace vlabmax = `vlmx' if varname == "`var'"
    }
}
```

```

/* fill in the sy#-variables for the survey years */

local nok " " /* list: inadmissible values of the wave indicator (blank is important) */

if `u'strlen("`syv'") == 0 { /* if there was no wave indicator in the file */
    local mark 0
    local c = `nw'
    forvalues j = 1(1)`c' {
        local scrpt = `j'-1+'wave1'
        replace sy`scrpt' = -1
    }
}

if `u'strlen("`syv'") > 0 { /* if there was a wave indicator in the file */
    local c : word count `sylev`k''
    forvalues j = 1(1)`c' {
        local sln = word("`sylev`k''", `j')
        local mark = real("`sln'")
        if `mark' >= `wave1' & `mark' <= `wave2' {
            capture confirm new variable sy`sln', exact
            if _rc != 0 {
                replace sy`sln' = 1 if varname == "`var'"
            }
        }
        if `mark' < `wave1' | `mark' > `wave2' { /* Cond. identical to _rc == 0 */
            if `u'strpos("`nok'", " `mark' ") == 0 {
                local nok "`nok' `mark'"
            }
        }
    }
    local sylev`k' "" /* emptied, otherwise information might be transferred to
                        processing of next file */
}

/* fill in the val#-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' "" /* emptied, otherwise information might be transferred to
                        processing of next file */
}

/* fill in the lab#-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'+1, .) /* rest until end of string */
        local a = `u'strpos("`str'", "")
        local x = `x' + 1
    }
    local labels`k' "" /* emptied, otherwise information might be transferred to
                        processing of next file */
}

} /* closing bracket for: quietly */
} /* closing bracket for loop over variables in allvars */

if `u'strlen("`nok'")' > 0 & "`nok'" != " " {
    local message " " _n ///
        "The wave indicator `syv' from file `datei' _n ///
        " contains the inadmissible value(s) `nok'"
    set output proc
    display "`message'"
    set output error
}
save "`pfad1d'/'`datei'_xxx`s'.dta", replace

if `s' == `zn' { /* In a last portion */
    if `zn' > 1 {
        local p = `zn'
        forvalues t = 1(1)`p' {

```

```

        use "`pfad1d'/'datei'_xxx`t'.dta", clear
        local nvr = `nvr' + nvr[1]
        local nvrnm = `nvrnm' + nvrnm[1]
    }
    local p = `zn'
    use "`pfad1d'/'datei'_xxx1.dta", clear
    forvalues t = 2(1)`p' {
        append using "`pfad1d'/'datei'_xxx`t'.dta"
    }
    /* diminishing by no. of surveyindicator-vars */
    local abz = (`zn'-1)*`nsyrl'
    quietly replace nvr = `nvr'-`abz'
    quietly replace nvrnm = `nvrnm'-`abz'
    duplicates drop
}
save "`pfad1d'/'datei'_meta`we'.dta", replace
/* delete portion files */
local p = `zn'
forvalues t = 1(1)`p' {
    capture erase "`pfad1d'/'datei'_xxx`t'.dta"
}
} /* closing bracket for: in a last portion */
} /* closing bracket for: generate single meta file */
} /* closing bracket for: processing of a file / portion */
} /* closing bracket for: if there are observations in the file */
} /* closing bracket for: if file exists */

} /* closing bracket for: loop over all data files */

if `u'strlen("`svd'") > 0{
    /* compile the single meta files */

    local nds : word count `svd'
    local a = `nds'
    if `a' > 1 {
        local message "      _n ///
        "The single meta files are compiled. " _n ///
        "      _n ///
        "Processing... Please wait. " _n ///
        "      "

        set output proc
        display "`message'"
        set output error
    }
}

```

```

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

/* modification of the variable undoc, if only missing codes und possibly value 0 are labeled */
replace undoc = -9 if undoc == nonmis & undoc != .
replace undoc = -9 if undoc == (nonmis - nvalzero) & undoc != .

/* modification of the val#- und lab# variables: ersatz values (missing codes) */
forvalues x= 1(1)100 {
    quietly {
        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
        replace lab`x' = "-200" if vallabset == "-2" & isnumeric == 1
        replace lab`x' = "-100" if isnumeric == 1 & vallabset != "-2" & uniqvalpos > 100
        replace lab`x' = "-100" if isnumeric == 0 & uniqvalpos > 100
        replace lab`x' = "-900" if isnumeric == 1 & vallabset != "-2" & uniqvalpos <= 100 & nonmis == 0
    }
}
/* generate consecutive number */
capture drop lfn
gen long lfn = _n
lab var lfn "consecutive number"
order lfn, first

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

/* erase single meta files */

local a = `nds'

forvalues i = 1(1)`a' {
    local datei = word("`svd'", `i')
    capture erase "`pfad1d'/'`datei'_meta`we'.dta"
}
} /* closing bracket for: compile single meta files */

```

```

/*#####
#####
#####          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"
    }

    if `u'strlen("`u'") > 0 {
        local message " " _n ///
        "Job completed. " _n ///
        "Runtime of the program: `hrs' hrs `min' min `sec' sec " _n ///
        "END at $$_TIME on $$_DATE " _n ///
        " " _n ///
        " Name of the generated meta file: " _n ///
        " `result' " _n ///
        " in directory `pfad1d' " _n ///
        " " _n ///
        " files that have not been processed: " _n ///
        " " _n ///
        "....- files not found: `fnexist' " _n ///
        " " _n ///
        "....- files with 0 observations: `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 $$_TIME on $$_DATE " _n ///
    " " _n ///
    " Name of the generated meta file: " _n ///
    "   `result' " _n ///
    "   in directory `pfad1d' " _n ///
    "   " _n ///
    " files that have not been processed: " _n ///
    "   " _n ///
    "....- not format of the used Stata-version: `s14file' " _n ///
    "   " _n ///
    "....- files not found: `fnexist' " _n ///
    "   " _n ///
    "....- files with 0 observations: `nobsfile' " _n ///
    "   "
  }
} /*closing bracket for: quietly */

set output proc
display " `message' " _n ///
      " `message1'"
capture log close

```