Slide 1

Transfering Data with *outdat.ado*

Ulrich Kohler, University of Mannheim, Germany

ukohler@sowi.uni-mannheim.de

May 23 2002

My presentation is about "outdat.ado", a program to transfer data from Stata to other statistical software packages. I know that there are already software packages like Stat/Transfer and that they do a pretty good job. I also know that it is a bad idea to transfer data from Stata to another package as we usually don't want to use another package than Stata to analyze data. So let me say something for ustification:

At first I like to state that there are at least some reasons for using other software than Stata. Some of my colleagues use Systat for graphs, for example, and I must admit that Systat here is in some ways better than Stata.

My second justification has to do with my position as a lecturer at Mannheim University. In Mannheim we use Stata as the main software. Sometimes our students want to share their data with others who working with SPSS. They come to me and ask what they can do. And I usually answer: "You can use that wonderful piece of software calling Stat/Transfer for this". And then they say: "Well we don't have that wonderful piece of software calling Stat/Transfer". And then I say: "Ok, I am doing this for you". So students go back to their computers, send me the data and then I translate the data and send the data back to the students. I really don't like that.

Both points seems not be too important. But while discussing outdat with my college Frauke Kreuter we realized two additional points, which we found much more compelling. Both points have to do with the way "outdat.ado" transfers the data. So let me first explain how `outdat` works and then explain these points afterwards.

Slide 2

Structure of Presentation

1. Demonstration
2. General Idea
3. How to Expand
4. Why?

I will start to show how `outdat` works by doing an example.

Then I will introduce the general idea of *outdat.ado*.

Then I will give an impression what needs to be done to expand *outdat.ado* to other software packages than those already implemented.

Finally I will talk about the two points, why a program like `outdat` may be useful in a broader sense.

Slide 3

Syntax-Diagram and Example

```
outdat [varlist] using filename
      [in range]
      [if exp]
      [, replace type(package)]

.  use data1
.  outdat gender area using data1, replace type(spss)
```

The Syntax of `outdat` is easy. The crucial part is the option `type()`. This option specifies the software package into which the data should be transfered. Unfortunately at the time being it is only possible to transfer to SPSS, Limdep and Stata. Note that Limdep is not tested as I only have had access to the manuals not to the software. You may even wonder about the silly thing to transfer data from Stata to Stata. But let us discuss thise after I made the two points at the end of presentation.

Let me try to use `outdat` for SPSS. Lets use some data first

```
.  use data1
```

and then try `outdat`:

```
.  outdat gender area using data1, replace type(spss)
```

We are specifying the package by mention its name in the `type()`-option.

As a result of the command you get two files: *data1.dat* and *data1.sps*. Lets take a look at them.

Slide 4

data1.dat

```
      1              1
      1              1
      2              .
      2              2
      1              2
      1              2
      1              2
      1              3
      1              3
      1              2
      1              1
      2              2
snip ✂
```

data1.dat is just a piece of ASCII data. To be more specific: it is ASCII data written by `outfile` with the `nolabel` option.

Now lets take a look at *data1.sps*.

Slide 5

data1.sps

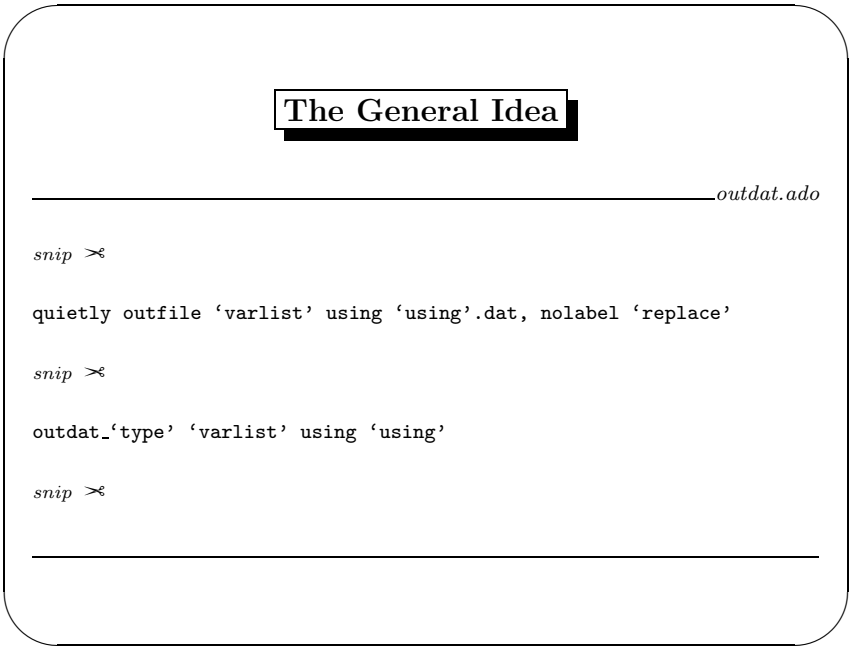
```
* SPSS-syntax to read and label data1.dat

DATA LIST FILE = "data1.dat" FREE / gender area .
VARIABLE LABELS
GENDER "Gender"
AREA "Neighborhodd" .
VALUE LABELS
GENDER
1 "men"
2 "women"
/AREA
1 "ancienty houses"
2 "new houses"
3 "mixed age"
4 "shops and houses"
5 "industrial area"
6 "other" .
exe.
```

data1.sps is a SPSS Syntax-File. It contains the command to read the data, the command to define the variable label and the command to define the value labels. I will call a file like this "dictionary" throughout this presentation. Note that in this specific dictionary there is a slash in the front of the second variable in the value label command, as SPSS wants it to be there. And yes, there is a point at the end of each command which SPSS really loves to have there.

Now, lets load the data into SPSS. This can be done by starting SPSS, loading *data1.sps* and run it. Note that SPSS gives some warning messages while running. This is, because SPSS doesn't understand that the points in *data1.dat* are missing values. But as SPSS doesn't understand the points at all, it assigns missing values to them, which is just fine.

Slide 6



The Basic Idea of *outdat.ado* is to split the process into two parts. The first part is to simply write the data into an ASCII-File.

```
quietly outfile 'varlist' using 'using'.dat, nolabel 'replace'
```

The second part is to create the dictionary. In *outdat.ado* this part is done with the command

```
outdat_'type' 'varlist' using 'using'
```

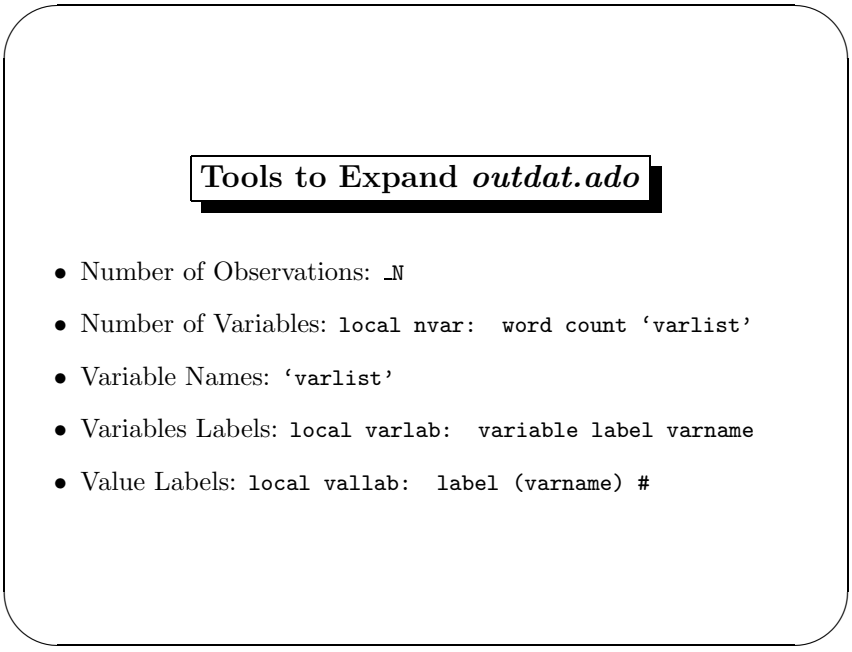
where 'type' is replaced with the name given in the type option. That is: `outdat` calls a subprogram which wrote the dictionary. Therefore, if there is an ado-file called "outdat_whatever.ado" somewhere around the path, we could specify `outdat` with the `type()` option "whatever" and `outdat` would call this Ado to write the dictionary.

Given, that all statistical software packages can read ASCII-Files, the first part of transferring Stata-data to another format always stays the same. We only need to worry about the second part.

It is possible to write modules for various software packages without changing *outdat.ado* at all. This is very much like the possibility to expand "egen" with self-made egen-functions. To add another outdat-type one has to write an ado calling `outdat_'whatever'`. For example to add an interface for SAS, one may write *outdat_sas.ado*.

To write such an ado isn't too hard. To do it one should rely onb the newly added `file` command. With this command one can write arbitrary text to arbitrary files within a Stata session. Therefore we can use this command to write software specific commands to a file.

Slide 7



Here you can see the list of informations one usually needs, to load ASCII-Data into a software package and the means to get this information in an ado.

Before calling the subprogram *outdat.ado* keeps only the cases included in *If*- or *In*-Conditions. Therefore one can get the number of cases with the build-in variable `_N`.

The variables to be transfered are passed as a varlist from *outdat.ado* to the subprogram. The number of words of that varlist equals the number of variables. The varlist also contains the names of the variables.

To get the variable labels for each variable of the varlist one has to loop over the varlist and catch the variable label with the extended macro function `variable label`.

Finally to get the value label for each variable one has to loop over each category of each variable and to catch the labels with the extended macro function `label (varname) #`.

Albeit it seems not possible to present a general framework for expanding `outdat`, it may be helpful to look at *outdat_spss.ado* as one example.

Slide 8

Syntax of *outdat_spss.ado*

`outdat` calls `outdat_spss` with:

```
. outdat_spss varlist using filename
```

`outdat_spss` catches the `varlist` and the `filename` with its own syntax-statement:

outdat_spss.ado

```
version 7.0
program define outdat_spss
syntax [varlist] using/
snip ≻≪
```

As you can see here `outdat` calls `outdat_spss` with `outdat_spss 'varlist' using filename` and `outdat_spss` caches the varlist and the filename with its own syntax-statement. Now lets see, how `outdat_spss` uses the varlist and the filename:

Slide 9

Building the Data-List

outdat_spss.ado

```
snip >8
file open spsfile using ‘using’.sps, replace text write
file write spsfile ‘"* SPSS-Syntax to read and label ‘using’.dat"’
file write spsfile  /*
*/ _n _n ‘"DATA LIST FILE = "‘using’.dat" FREE / ‘varlist’ ."’
snip >8
```

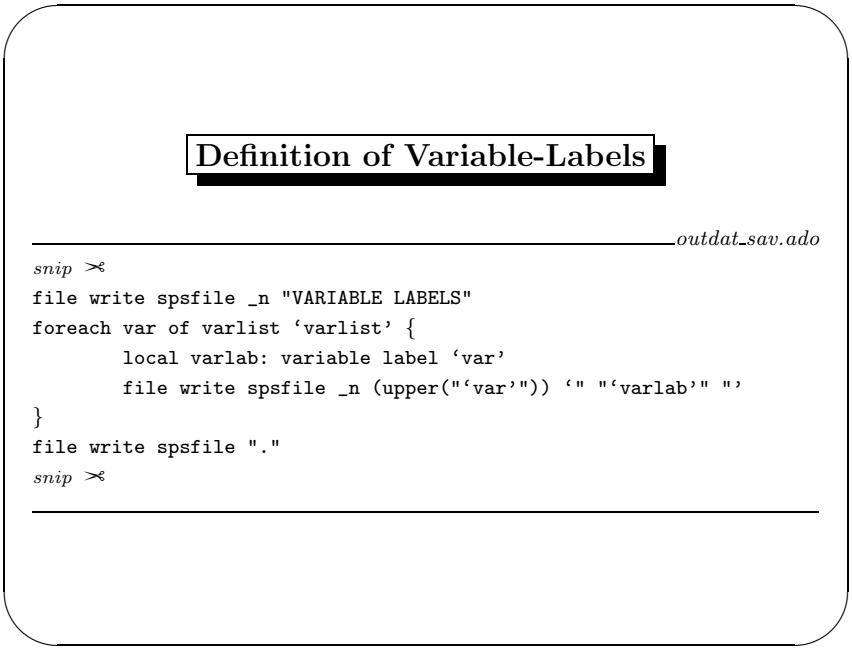
We first use `file` to open a text file with the name specified by the using-part of `ouddat` and the extension “sps”. This file is going to be the data dictionary. SPS happens to be the standard extension of SPSS-Syntax-Files, so I decided to hard-code this extension to “outdat_spss”.

Afterwards we write a comment line to have a nice caption in the Dictionary-File.

The next command first adds two new lines and then wrote the SPSS-command `Data List File = using.dat free / varlist` into the growing dictionary. This is the SPSS command to read ASCII-Files. The command requires to specify variable names. I put them into the Data-List command with the local macro *varlist*, which expands to the variable names.

Now lets turn to the definition of variable-labels.

Slide 10



We start by writing the command “Variable labels” to the Dictionary. Then we loop over each variable of the varlist by using `foreach`—the Stata 7 way of saying `while`. Inside the loop we catch the variable label and write down this variable label in SPSS-Syntax to the sps-file.

Finally let me show you the crucial part of the code for the value labels:

Slide 11

Definition of Value-Labels

outdat_sav.ado

```
snip ≻≪
if ‘i’ == 1 {
  file write spsfile _n (upper("‘var’"))
  local i = ‘i’ +1
}
else {
  file write spsfile _n "/" (upper("‘var’"))
}
forvalues j = 1/‘K’ {
  summarize ‘var’ if ‘kvar’ == ‘j’, meanonly
  local k = r(mean)
  local vallab: label (‘var’) ‘k’
  if "‘k’"  ~= "." {
    file write spsfile _n ‘"‘k’ "‘vallab’" "’
  }
}
```

This is slightly more complicated. The part you can see here is inside a loop over the variables. Before starting the loop I have already wrote the SPSS-Command “VALUE LABEL” to the dictionary.

Inside the loop the first step is to write the name of the variable—without leading slash for the first name and with leading slash for any other names.

Afterwards I start to loop over the values of the variable and write the label attached to the value in question in SPSS-Syntax to the dictionary.

There is an additional complication in doing this as there may be gaps between the values. My solution for this is to construct a variable which numbers the categories of the variable in question. I summarize the variable in question for each category of this variable and use the mean to find my value label. If there are better solutions for this, I would be happy to hear about them.

Finally I have to answer the main question: Why do we need yet another transfer program.

Slide 12

Why?

- The dictionary-files produced with `outdat` can be easily edited to fit ones personal requirements
- It is better to archive data as ASCII-Files together with an dictionary than as system-files

As I said before I have two points to make here. The first one is that the dictionary files produced with `outdat` can be easily edited to fit ones personal requirements. This way one can most easily make different dictionaries for different purposes.

My second justification is a more general one. I believe that reproducibility should be a main feature of data analysis. That's why I think one should archive data. Archiving means archiving it for perpetuity and therefore one should not archive system-files. Do you know what happens with your Stata data-sets in—say—30 years? Will Stata Corp still exist? Will there still be a computer program which can read Stata System Files? I hope so, but I won't bet on it. Instead I think we should archive data in a form which is as much human-readable as possible. Therefore we should archive data-sets as ASCII-Files together with a complete description of the data. One way to design the complete description are the software specific commands to read and label the ASCII-Files: the dictionaries. If we store dictionaries for several software packages our data stays readily accessible for users of each of the software packages. The storing of the data would be parsimonious, as one need to store the data only once and dictionaries are small. And our data set would be accessible forever as historians of coming centuries should be wise enough to understand the logic of the dictionaries. Mankind have managed to understand the hyroglyphes so they should manage to understand some SPSS commands, don't they.

I will send *outdat.ado* to Kit Baum shortly, but not before I have implemented your comments, if any. Thank you for listening.