The Stata Journal

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press, and Stata is a registered trademark of StataCorp LP.

The Stata Journal (2005)
5, Number 2, pp. 162–187

Multilingual datasets

Jeroen Weesie
Utrecht University

Abstract. This insert describes a new command `mlanguage` that facilitates the creation and maintenance of "comprehensive" multilingual datasets. These are datasets with many variables, many of which are value labeled, with labels in different languages, all contained within the dataset. The tools make it easy to add labels in a new language by translating an existing set of labels, to switch between the sets of labels, to verify the integrity of such labels, and to assist in keeping the labels complete.

Keywords: dm0013, mlanguage, multilingual datasets, data integrity, value labels

1 Introduction

The 9 September 2003 update of Stata 8 introduced the ability to have up to 100 different sets of data, variable, and value labels in a dataset. A dataset might contain one label set in English, another in German, and a third in Dutch. While a dataset may contain multiple sets of labels, one set of labels will be in use at any one time. Switching between these sets is easy and fast. When other Stata commands produce output (such as `describe`, `tabulate`, `codebook`, etc.), they use the labels of the active (currently set) language. Other aspects of the output, such as the table headers, the online help, etc., are of course not affected—they are always in English. When you define or modify the labels using the other `label` commands, such as `label variable` or `label value`, you modify the active (current) set of labels. The different sets of labels are automatically saved with your data. Moreover, when you `use` the data the next time, Stata will remember what language you selected last before saving the data.

The different sets of labels are called "languages", reflecting their most likely application representing different spoken languages; you do not need to use the multiple sets in this way. Another useful application would be to create a dataset with one set of long labels and another set of shorter ones, or you could temporarily switch off labeling in output; however, dropping label information involves a permanent loss of information. If you define a language with an empty set of labels, switching between the original and the empty language switches the display of labels on and off.

Since readers of the *Stata Journal* may not yet be familiar with this recently added functionality of `label`, I give a brief description in section 2. In section 3, I introduce `mlanguage`, a new command that was written to facilitate the creation and maintenance of "big" multilingual datasets. For instance, `mlanguage` makes it easy to add a label set in a new language to a dataset that is already labeled. Think of this as translating a collection of strings from one language into another language. The main work has to be done by a person who can translate in a text file a set of label strings from, say, German

into English; this person need not know Stata. Other tools facilitate the verification that the labels are consistent, complete, and compatible across languages (in a sense described below) and help to restore data integrity if problems are encountered.

2 The language subcommand of label

In this section, I briefly describe the main features of the command `label language`. For more information, refer to the online help (make sure that you update Stata so that it is available); if you already own Stata 9, see [D] **label language**. `label language` provides subcommands for selecting a language (set of labels), defining a new empty language, copying a language, renaming a language, and dropping a language.

2.1 Syntax

`label language`

`label language` *languagename*

`label language` *languagename*`, new [copy]`

`label language` *languagename*`, rename`

`label language` *languagename*`, delete`

2.2 Description

`label language` (without arguments)

lists the available languages and the name of the active one. The active or current language refers to the labels you will see if you use, say, `describe` or `tabulate`. The available languages refer to the names of other sets of previously created labels. For instance, you might currently be using the labels in `en` (English), but also available might be labels in `de` (German) and `nl` (Dutch).

`label language` *languagename*

changes the labels to those of the specified language. For instance, if `label language` revealed that `en`, `de`, and `nl` were available, typing `label language de` would change the current language to `de`.

`label language` *languagename*`, new` [`copy`]

creates a new set of labels collectively named *languagename*. You may name the set as you please as long as the name does not exceed 8 characters. For suggestions on naming spoken languages, refer to section 4. Initially all labels are empty unless you specify the option `copy`, which initializes the labels to those of the current language.

`label language` *languagename*`, rename`

changes the name of the label set currently in use. If the label set in use were named `default` and you now wanted to change that to `en`, you could type `label language en, rename`.

The choice of the name `default` in the example was not accidental. If you have not yet used `label language` to create a new language, the dataset will have one language that will be named `default`.

`label language` *languagename*`, delete`

deletes the specified label set. If *languagename* is also the current language, one of the other available languages is chosen to become the current language. You should explicitly select the language you want to be active after dropping a language with `label language` *languagename*.

2.3 A first application

Often I want to switch between looking at numerical data and looking at (value) labeled data. Think of doing an analysis for females only, based on the value-labeled variable `sex`. We need to use the expression `if sex==?`, with `?` denoting the females. What numerical value `?` was used in this dataset? Looking this up is somewhat awkward. Commands such as `tabulate` can suppress the labels, whereas other commands, such as `table` and `tabstat`, do not. What can we do? Dropping the labels requires a lot of work and cannot be undone easily. One possibility is to create "smart value labels" that contain both the numerical values and the category descriptions. The command `numlabel`, introduced in Stata 8, makes this easy because it allows you to modify existing value labels. For instance, the value labels

```
0   "female"
1   "male"
```

can be modified to

```
0   "[0]  female"
1   "[1]  male"
```

Output with such labels looks somewhat ugly, though, and Stata may have to display truncated labels. An alternative makes use of the multiple languages mechanism. Think of the set of empty labels as a language. Type the two statements

```
. label language full, rename
. label language null, new
```

to use the name `full` for the current set of labels and `null` for the empty set. Now if we want to look at labeled output, we make sure that the language `full` is selected

```
. label language full
```

while Stata will generate nonlabeled output if the `null` language is active:

```
. label language null
```

If you think this is too much to type, you can easily write two commands `labon` and `laboff` to switch the display of labels on and off.

```
program labon
        version 8.1, born(09sep2003)

        quietly label language full
        dis as txt "(labels will be displayed)"
end

program laboff
        version 8.1, born(09sep2003)

        quietly label language null
        dis as txt "(labels will not be displayed)"
end
```

2.4 Remarks

To create and work with a multilingual dataset, follow these steps:

1. define a first set of labels using `label data`, `label variable`, `label define`, and `label value`
2. optionally rename the first language from `default` to a desired name:
   `label language` *languagename*$_1$`, rename`
3. define a new language, which is initially empty:
   `label language` *languagename*$_2$`, new`
4. define labels in the new language, using the same commands as in step 1
5. repeat steps 3 and 4 as often as needed
6. save the data with all labels in all defined languages
7. activate the language you want to use
   - look at output in the labels in the activated language
   - modify the activated language

3 The mlanguage command

The main purpose of this article is to introduce a new utility for producing and managing comprehensive multilingual datasets. Being written "on top of" `label language`, `mlanguage` requires that your Stata version be no older than 9 September 2003 so that the `label language` subcommand is available. `mlanguage` offers facilities to make it easy to add a full set of labels in a new language by translating an existing set of labels, where the actual translation (e.g., from German to English) can be performed *outside* of Stata, possibly even by a person not well versed in Stata. Second, `mlanguage` offers tools for maintaining a series of comprehensive sets of labels, guarding that the labels across the languages satisfy some reasonable properties (see below for details).

`mlanguage` sometimes has to "propose" new value labels. These have to be named. `mlanguage` adopts a naming scheme for the value labels in different languages, namely, *basename_languagename*. For example, in a dataset with the labels in the languages `en`, `nl`, and `de`, the different versions of a value label `repair` are named `repair_en`, `repair_nl`, and `repair_de`. The language extension may actually be absent in one language. This may be a desirable situation if one of the languages is to be treated as a "base" language, e.g., the language in which the data were collected. If you prefer to treat all languages equally, you may want to change the existing value label names to match the scheme. The subcommand `mlanguage rename` *languagename*`, label` can be used for this purpose. While `mlanguage` does not enforce this naming convention, I suggest that you follow it because transparent and consistent naming reduces the likelihood of mistakes.

`mlanguage`, which may be abbreviated as `mlang`, is designed as a command with eight subcommands. This design allows the extension of the command with new types of functionality. Suggestions for such extensions are welcomed by the author.

3.1 Syntax

`mlanguage` [`dir`|`query`]

`mlanguage` {`select`|`set`} *languagename* [`:` *cmd*]

`mlanguage` {`drop`|`delete`} *languagename*$_1$ [`, select(`*languagename*$_2$`)`]

`mlanguage rename` *newlanguagename* [`, label`]

`mlanguage list` [*languagenamelist*] [`, nopattern nodescribe noseparator novalue varlist(`*varlist*`)`]

`mlanguage add` *newlanguagename*`, saving(`*filename*`)` [`nocomment noinstruction column(`#`) unlabeled varlist(`*varlist*`) replace`]

`mlanguage check` [*languagenamelist*] [`, varlist(`*varlist*`) same`]

`mlanguage fix` [*languagenamelist*]`, saving(`*filename*`)` [`, nocomment noinstruction column(`#`) novalue unlabeled varlist(`*varlist*`) replace`]

3.2 Description

I will describe the subcommands of `mlanguage` in some detail. The first four subcommands are wrappers for the corresponding `label language` commands. They are provided to offer a consistent interface while offering some minor additional functionality to `label language`.

`mlanguage` [`dir`|`query`]

displays the available languages and the name of the current language. The current language refers to the labels you would see if you used, say, `describe` or `codebook`.

The `dir` and `query` subcommands of `mlanguage` resemble `label language, dir` but produce less output. Typing `mlanguage` with no options is equivalent to typing `mlanguage dir`.

`mlanguage` {`set`|`select`} *languagename*

changes the labels to those of the specified language. For instance, if `mlanguage dir` revealed that `en`, `de`, and `nl` were available, typing `mlanguage select de` would change the current language to `de`.

The prefix syntax `mlanguage set`|`select` *languagename*`:` *cmd* runs *cmd* with labels in the language *languagename* but does not select *languagename* as the active language.

The command `mlanguage select` *languagename* is equivalent to `label language` *languagename*.

`mlanguage` {`drop`|`delete`} $languagename_1$ [`, select(`$languagename_2$`)`]

deletes label set $languagename_1$. If $languagename_1$ is also the current language, one of the other available languages is chosen to become the current language; the option `select()` selects $languagename_2$ to become the current one.

The command `mlanguage drop` *languagename* is equivalent to `label language` *languagename*`, delete`.

`mlanguage rename` *newlanguagename* [`, label`]

changes the name of the label set currently in use. If the label set in use were named `default` and you now wanted to change that to `en`, you could type `mlanguage rename en`. The name `default` in this example was not accidental. If you have not yet created a new language, the dataset will have one language that will be named `default`.

The command `mlanguage rename` *languagename* (without the option `label`) is equivalent to `label language` *languagename*`, rename`. The option `label` modifies the names of the value labels to match the naming convention adopted by `mlanguage`. It appends to the value labels of the active language the string *_languagename*. If the value labels in the active language already have a suffix, it is replaced by the string *_languagename*. Anyway, the links between variables and value labels are, of course, adjusted accordingly.

The other four subcommands of `mlanguage` offer facilities that are not matched by the `label language` subcommands.

`mlanguage list` [*languagenamelist*] ...

displays the label information (data label, variable labels, and value labels) for the specified languages. If no *languagenamelist* is specified, the information is displayed for all defined languages.

The data and variable labels are displayed in a `describe`-like format.

The value labels are organized by "language-label pattern": a collection of value labels (including "none") attached to one or more variables in different languages. For instance, if variables x_1, x_2, ..., use value label Lab_1 in language L_1, label Lab_2 in language L_2, etc., (L_1:Lab_1) (L_2:Lab_2) ... is called the language-label pattern used by the variables x_1, x_2, etc. In a typical application, such a collection of value labels is expected to contain translations in the languages L_1, L_2, ... This organization of the output makes it easy to verify translations.

`mlanguage add` *languagename* ...

assists in adding a collection of labels in a new language to the dataset in memory that has already been labeled in one or more languages. The procedure involves translating the labels from a *source* (= current) language into a *target* (= *languagename*) language. Three steps must be taken:

1. Make sure that the data are in memory and that the *source* language is selected as the current language. Invoke

   `mlanguage add` *target*`, saving(`*fn*`)`

   to generate a script file *fn*`.do` with a possibly long series of `label` commands that (i) define the new language *target*, (ii) define the data label, (iii) define the variable labels, (iv) define a new set of value labels, and (v) attach the new value

labels to the variables. The initial values of the labels are those of the *source* (current) language. The names of the new value labels defined in the script follow the naming scheme explained above; if the value label in the source language has the suffix _*source*, this suffix is replaced by _*target*; otherwise, _*target* is appended.

2. The literal strings in *fn*`.do` must be translated into the *target* language, probably by a human expert in the *target* language who need not be a Stata user. To facilitate verification of the translation, the *source* labels are included in comments, so they remain visible after the label texts are replaced by a translation. Be sure not to damage the Stata syntax itself. I advise you to do the translations in a copy of the file so that you can repair the Stata commands if you make a mistake.

3. When you have finished the translation, the translated file *fn*`.do` must be `run` with the dataset in memory; it does not matter which language is selected at this point. Only at this step are labels in the *target* language added to the dataset. Moreover, the language *target* is selected upon completion.

This process can be repeated as often as needed. The maximum number of languages in a dataset is 100. This is probably enough for all foreseeable applications, the possible exception being datasets of the European Union.

`mlanguage check` [*languagenamelist*] ...

verifies that the value labels are consistent, complete, and compatible across the languages in *languagenamelist*, or across all languages in the dataset if no such list is specified.

By consistency of value labels, I mean that if variables share a value label in some language, these variables cannot be attached to different value labels in another language. For instance, if the variables `edu_father` and `edu_mother` are both labeled by the value label `edu_en` in English, in German the value label `edu_de` cannot be attached to `edu_father` and `edu2_de` to `edu_mother`. This situation is inconsistent, even if `edu_de` and `edu2_de` are identically defined; see the utility command `labeldups` described in Weesie (2005) to find and eliminate duplicate value labels.

Completeness of the value labeling means that if a variable is value valued in one language, it is value labeled in other languages as well. If in German a value label is attached to `edu_father` while it is unlabeled in Dutch, the value labeling is said to be incomplete. Incompleteness of value labels, and of variable and data labels as well, can be fixed using `mlanguage fix`.

Compatibility means that the value labels attached to a variable in different languages provide mappings for the same set of values. If in English, mappings are provided for 0 (no) and 1 (yes), it would be strange (incompatible) if, in addition, in Spanish the value 2 meant “maybe”. This looks like a real mistake that you will have to fix manually.

Note that the command does *not* check that mappings are provided for all numerical values of the variables to which the labels are attached. See [D] **codebook** and [D] **labelbook** for such additional checks.

`mlanguage fix` ...

assists in making label information complete over all languages. Completeness of the data label, variable labels, and value labels means that if a label is available in one language, it is available in other languages as well. Thus completeness does not mean that "everything" should be labeled; see also option `unlabeled`.

Making value labels complete is only possible if they are consistent; see `mlanguage check` for details.

If the label information is found to be incomplete, `mlanguage fix` generates script files, one for each language with missing labels, with definitions in one of the other languages. The labels in these files, named *saving_languagename*`.do`, must be translated into the respective language *languagename*, and finally, these do-files must be executed.

3.3 Options

`select(`*languagename*$_2$`)`, an option with subcommands `drop` and `delete`, specifies the language to become active after dropping language *languagename*$_1$.

`label`, an option with the subcommand `rename`, specifies that the names of the value labels be modified to satisfy the naming convention of `mlanguage`, namely, from *oldname* into *oldname_languagename*. This option may be specified only if the data are monolingual.

`nopattern`, an option with the subcommand `list`, suppresses the display of the value labels by "language-label pattern". `novalue` is a synonym when used with subcommand `list`.

`nodescribe`, an option with the subcommand `list`, suppresses the `describe`-like table, listing for each variable the variable and value labels in the different languages.

`noseparator`, an option with the subcommand `list`, suppresses the separator lines in the `describe`-like table with variable labels in the different languages.

`varlist(`*varlist*`)`, an option with the subcommands `add`, `check`, `fix`, and `list`, specifies the list of variables used in the subcommand. The default is to use all variables.

`saving(`*filename*`)` is required with the subcommand `add`. *filename* specifies the name of the file to be created. If no extension is specified, the extension `.do` is appended.

`saving(`*filename*`)` is required with the subcommand `fix`. *filename* should not contain an extension. For each of the specified languages *ln*, definitions of label information are included in the file *filename_ln*`.do`. For instance, if the dataset contains label information in the languages `en`, `nl`, and `de`, `mlanguage fix, file(todo)` creates

the files `todo_en.do`, `todo_nl.do`, and `todo_de.do` with label information to be translated into `en`, `nl`, and `de`, respectively. A language file will not be produced for a language for which no action is needed. A language file may consist of only a series of `label value` statements to attach existing value labels to variables; such files need not be edited and can be run unchanged. Usually, however, translations must be provided in these language files.

`nocomment`, an option with the subcommands `add` and `fix` only, suppresses including each text to be translated as a comment to facilitate verifying that translations are correct.

`noinstruction`, an option with the subcommands `add` and `fix` only, suppresses the instructions at the beginning of script files.

`novalue`, an option with the subcommands `list` and `fix` only, suppresses generating label statements that define or attach value labels.

`column(#)`, an option with the subcommands `add` and `fix` only, is rarely used. It specifies the column at which comments are to be written. The default is `column(60)`.

`unlabeled`, an option with the subcommands `add` and `fix` only, specifies that `label` statements be generated for the data label and for the variable labels that are currently undefined.

`replace`, an option with the subcommands `add` and `fix` only, specifies that output files be overwritten if they already exist.

`same`, an option with subcommand `check` only, verifies that value labels attached to a variable for different languages provide mappings for the same set of values ("compatibility").

4 ISO-639 language codes

You may name languages as you please. You may name Dutch labels `Nederlnd`, `Holland`, `Dutch`, `LowLands`, or whatever else appeals to you. `label language` and `mlanguage` allow language names of up to 8 letters; the names may not contain nonalphabetic characters. The language names `Nederlands` (too long) and `Pays-Bas` (invalid character) are therefore not allowed.

For consistency across datasets, if the language you are creating is a spoken language, I recommend that you use the ISO 639-1 two-letter codes if possible; ISO-639 provides three-letter codes for less-widely used languages. A subset of the codes is listed in the table below.

(*Continued on next page*)

alpha2	alpha3	English name of Language	alpha2	alpha3	English name of Language
ar	ara	Arabic	it	ita	Italian
	bnt	Bantu	ja	jpn	Japanese
eu	baq	Basque	la	lat	Latin
bg	bul	Bulgarian	lv	lav	Latvian
zh	chi	Chinese	lt	lit	Lithuanian
hr	scr	Croation	no	nor	Norwegian
cs	cze	Czech		pap	Papiamento
da	dan	Danish	fa	per	Persian
nl	dut	Dutch and Flemish	pl	pol	Polish
en	eng	English	pt	por	Portuguese
eo	epo	Esperanto	ro	rum	Romanian
et	est	Estonian	ru	rus	Russian
fi	fin	Finnish	sr	scc	Serbian
fr	fre	French		sgn	Sign languages
fy	fry	Frisian	sk	slo	Slovak
de	ger	German	es	spa	Spanish; Castilian
el	gre	Greek (modern)	sw	swa	Swahili
kl	kal	Greenlandic	sv	swe	Swedish
he	heb	Hebrew	th	tha	Thai
hi	hin	Hindi	tr	tur	Turkish
hu	hun	Hungarian	uk	ukr	Ukrainian
is	ice	Icelandic	uz	uzb	Uzbek
id	ind	Indonesian	vi	vie	Vietnamese
ga	gle	Irish	cy	wel	Welsh

The full list can be found at

http://lcweb.loc.gov/standards/iso639-2/iso639jac.html

5 Example

As an illustration of how `mlanguage` can assist you in the construction and maintenance of a multilingual dataset, I consider the automobile data often used in Stata documentation. For simplicity, I have only kept the variables `make`, `price`, `rep78`, `rep79`, and `foreign`.

(*Continued on next page*)

```
. describe
Contains data from cardata.dta
  obs:            74
 vars:             5                          18 Apr 2005 16:32
 size:         1,924 (99.9% of memory free)    (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str17  %-17s
price           int    %8.0gc
rep78           byte   %13.0g
rep79           byte   %9.0g
foreign         byte   %8.0g
-------------------------------------------------------------------------------
Sorted by:  foreign
```

The output is fairly incomprehensible, especially if you are unfamiliar with the data. The dataset is not labeled; the variables are not labeled; and moreover, the categorical variables `rep78`, `rep79`, and `foreign` are not value labeled. In fact, I removed the labels to show how to add labels.

5.1 Adding labels in a first language

Three pieces of label information can be added to this dataset. First, I add a label describing the dataset:

```
. label data "1978 Automobile Data"
```

Next, I add descriptive labels to the variables:

```
. label var make     "Make and Model"
. label var price    "Price"
. label var rep78    "Repair Record 1978"
. label var rep79    "Repair Record 1979"
. label var foreign  "Car type"
```

Finally, I want to add value labels describing the categories of variables. Stata treats value labels as special objects that are defined once and can be attached to as many variables as needed.

```
. label define repair 1 "very bad" 2 "bad" 3 "reasonable" 4 "good" 5 "very good"
. label value rep78 repair
. label value rep79 repair
. label define origin 0 Domestic 1 Foreign
. label value foreign origin
```

Now the output of `describe` and `label list` looks more appealing:

```
. des
Contains data from cardata.dta
  obs:            74                          1978 Automobile Data
 vars:             5                          18 Apr 2005 16:32
 size:         1,924 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str17  %-17s                  Make and Model
price           int    %8.0gc                 Price
rep78           byte   %13.0g      repair     Repair Record 1978
rep79           byte   %10.0g      repair     Repair Record 1979
foreign         byte   %8.0g       origin     Car type
-------------------------------------------------------------------------------
Sorted by:  foreign
. label list
origin:
           0 Domestic
           1 Foreign
repair:
           1 very bad
           2 bad
           3 reasonable
           4 good
           5 very good
```

at least if you prefer to look at labeling information in English. I have now defined a single set of labels. Like `label language`, `mlanguage` refers to different sets of labels as *languages*. The subcommand `dir` (or its synonym `query`, and in fact also the "empty" subcommand) displays the available languages and the currently active language.

```
. mlanguage dir
Value and variable labels have been defined in only one language: default
```

We are told that label information is available in only one language. This language has not yet been identified and is therefore referred to as `default`. I prefer to make it clear that the labels that I entered above are the English versions, so I use the two letter (alpha2) ISO-639 code `en` for English. The `rename` subcommand of `mlanguage` can be used to change the language name from the current value `default` to `en`.

```
. mlanguage rename en, label
(language default renamed en)
. mlanguage dir
Value and variable labels have been defined in only one language: en
```

The option `label` of `mlanguage rename` specifies that the names of the value labels be renamed to fit the naming scheme for value labels in multilingual datasets adopted by `mlanguage`, namely, *basename_languagename*. Below, I will add Dutch (`nl`) and German (`de`) versions of value labels. At that moment, there will be three versions of the `repair` value label:

`repair_en`	English version of value label `repair`
`repair_nl`	Dutch version of value label `repair`
`repair_de`	German version of value label `repair`

The option `label` ensured that the English versions of the value labels are called `repair_en` and `origin_en` rather than `repair` and `origin` so that all languages are treated equally. After this last modification, the English labeling in the dataset is complete.

```
. describe
Contains data from cardata.dta
  obs:            74                          1978 Automobile Data
 vars:             5                          18 Apr 2005 16:32
 size:         1,924 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str17  %-17s                  Make and Model
price           int    %8.0gc                 Price
rep78           byte   %13.0g      repair_en
                                               Repair Record 1978
rep79           byte   %10.0g      repair_en
                                               Repair Record 1979
foreign         byte   %8.0g       origin_en
                                               Car type
-------------------------------------------------------------------------------
Sorted by:  foreign
```

The names of the variable labels in the output look somewhat poorly aligned. This is due to the larger length of the names of the value labels.

5.2 Adding labels in a second language

Next I want to add support for the Dutch language with ISO-639 alpha2 code `nl`. This requires a large number of `label` statements, paralleling those that defined the English labels.

```
. label language nl, new
. label data "Dutch data label"
. label var rep78 "Dutch label for rep78"
  ...
. label define repair_nl 1 "text1" 2 "text2" ... 5 "text5"
. label value rep78 repair_nl
. label value rep79 repair_nl
  ...
```

This is just a pet example, as I speak Dutch myself. Therefore, this is not very difficult, and I should be able to do this without error. With a dataset with thousands of variables and value labels, it is hardly feasible to work in this way. Moreover, if I wanted to produce a version in Spanish, I would have to turn to a translator. The subcommand `add` of `mlanguage` is designed for this situation. I will first add labels in the language `nl`:

```
. mlanguage add nl, saving(tonl) column(55)
file tonl.do was successfully created
```

`mlanguage` created a text file `tonl.do` with the required Stata `label` commands. Note that these commands have not yet been processed. We must edit the file before executing the commands.

```
. type tonl.do
// instruction
// (1) translate the quoted strings into language nl
// (2) save the file under a new name
// (3) execute the saved file with the current data set in memory
// (4) use -mlanguage list- to verify results
label language nl, new
label data `"1978 Automobile Data"'                    // 1978 Automobile Data
label var make `"Make and Model"'                      // Make and Model
label var price `"Price"'                              // Price
label var rep78 `"Repair Record 1978"'                 // Repair Record 1978
label var rep79 `"Repair Record 1979"'                 // Repair Record 1979
label var foreign `"Car type"'                         // Car type
label define origin_nl 0  `"Domestic"'                 // Domestic
label define origin_nl 1  `"Foreign"' , add            // Foreign
label define repair_nl 1  `"very bad"'                 // very bad
label define repair_nl 2  `"bad"' , add                // bad
label define repair_nl 3  `"reasonable"' , add         // reasonable
label define repair_nl 4  `"good"' , add               // good
label define repair_nl 5  `"very good"' , add          // very good
// no changed needed after this point
label value rep78 repair_nl
label value rep79 repair_nl
label value foreign origin_nl
// end-of file
```

The file starts with a series of comment lines with instruction; recall that Stata treats all text after `//` until the end-of-line as a comment. The first statement

```
label language nl, new
```

adds a new language, `nl`, initially without any labels. The rest of this file labels the data: there is one command line for the description of the dataset, five command lines with variable labels, two command lines for the definition of the value label `origin_nl`, and five lines for `repair_nl`. The value labels are defined one mapping at a time, making it easier to work in the file. It would also be necessary if we were dealing with value labels with lots of mappings. In quoted strings, the *English* labels are given—Stata cannot, of course, do the translation itself. These strings should be translated into their Dutch equivalents. Throughout compound quotes `` `" `` and `"' `` rather than the simpler double quotes `"` are used; compound quotes allow quotes in the label texts. The file contains the English labels in comments. These comments should not be changed. Including the original labels in comments facilitates the translation and verification process.

In this case, I translated the English labels in the file `tonl.do` into their Dutch equivalents and named this file `addnl.do`. The contents of this file are

```
. type addnl.do
// instruction
// (1) translate the quoted strings into language nl
// (2) save the file under a new name
// (3) execute the saved file with the current data set in memory
// (4) use -language list- to verify results

label language nl, new

label data `"Gegevens over personenauto's (1978)"'     // 1978 Automobile Data
label var make `"Merk en model"'                       // Make and Model
label var price `"Prijs"'                              // Price
label var rep78 `"Onderhoud 1978"'                     // Repair Record 1978
label var foreign `"Auto type"'                        // Car type
label var rep79 `"Onderhoud 1979"'                     // Repair Record 1979

label define origin_nl 0  `"Amerikaans"'               // Domestic
label define origin_nl 1  `"Overig"' , add             // Foreign

label define repair_nl 1  `"zeer slecht"'              // very bad
label define repair_nl 2  `"slecht"' , add             // bad
label define repair_nl 3  `"redelijk"' , add           // reasonable
label define repair_nl 4  `"goed"' , add               // good
label define repair_nl 5  `"zeer goed"' , add          // very good

// no changed needed after this point
label value rep78 repair_nl
label value foreign origin_nl
label value rep79 repair_nl

// end-of file
```

We are now ready to actually add the Dutch labels to the data. All we have to do is to execute `addnl.do`, using `do` or `run`, making sure that the original data are in memory:

```
. do addnl
```

Indeed the dataset is now bilingual, with `nl` the active language.

(*Continued on next page*)

```
. mlanguage
Available languages : en nl
   Current language : nl
. describe
Contains data from cardata.dta
  obs:            74                          Gegevens over personenauto's
                                                (1978)
 vars:             5                          18 Apr 2005 16:32
 size:         1,924 (99.9% of memory free)   (_dta has notes)
```

variable name	storage type	display format	value label	variable label
make	str17	%-17s		Merk en model
price	int	%8.0gc		Prijs
rep78	byte	%13.0g	repair_nl	Onderhoud 1978
rep79	byte	%11.0g	repair_nl	Onderhoud 1979
foreign	byte	%10.0g	origin_nl	Auto type

```
Sorted by:  foreign
. label list
repair_nl:
           1 zeer slecht
           2 slecht
           3 redelijk
           4 goed
           5 zeer goed
origin_nl:
           0 Amerikaans
           1 Overig
origin_en:
           0 Domestic
           1 Foreign
repair_en:
           1 very bad
           2 bad
           3 reasonable
           4 good
           5 very good
```

You can now `tabulate` a variable with Dutch value labels:

```
. tab rep78
```

Onderhoud 1978	Freq.	Percent	Cum.
zeer slecht	2	2.90	2.90
slecht	8	11.59	14.49
redelijk	30	43.48	57.97
goed	18	26.09	84.06
zeer goed	11	15.94	100.00
Total	69	100.00	

The English labels are still available—all sets of labels are saved with the data. We can switch easily, and almost instantly, between languages:

```
. mlanguage select en
. tab rep78
```

Repair Record 1978	Freq.	Percent	Cum.
very bad	2	2.90	2.90
bad	8	11.59	14.49
reasonable	30	43.48	57.97
good	18	26.09	84.06
very good	11	15.94	100.00
Total	69	100.00	

The English labels were activated and are currently active. You can also use a set of labels temporarily.

```
. mlang set nl: tab rep78 rep79
  (output omitted)
```

5.3 Adding more languages

We continue adding a third set of labels in German with ISO code `de`. `mlanguage add` creates a file with the labels in the "current language". If you prefer to translate from English to German, you must make sure that English is the current label language:

```
. mlanguage select en
. mlanguage add de, saving(tode)
```

Alternatively, if you prefer translating Dutch into German, you should make Dutch the current language before invoking `mlanguage add`.

```
. mlanguage select nl
. mlanguage add de, saving(tode)
```

The value labels that will contain the German labels are `origin_de` and `repair_de`. The "initial" label texts are in Dutch because Dutch was the current language at the time the file `tode.do` was created. Next we must translate the Dutch labels into German and save the edited file as `addde.do`. To save paper, we do not show the contents of `tode.do` and `addde.do`. Finally, to add `de` labels, the translated script `addde.do` must be executed.

```
. do addde
```

This results in a trilingual dataset:

```
. mlang
Available languages : de en nl
   Current language : de
```

We can obtain a comparative view of the labeling information using the `list` subcommand of `mlanguage`:

```
. mlanguage list en nl de
Label information for languages en nl de
Contains data from  cardata.dta
                en  1978 Automobile Data
                nl  Gegevens over personenauto's (1978)
                de  Autodaten 1978
```

Variable	Language	Value label	Variable label
make	en		Make and Model
	nl		Merk en model
	de		Marke und Modell
price	en		Price
	nl		Prijs
	de		Prijs
rep78	en	repair_en	Repair Record 1978
	nl	repair_nl	Onderhoud 1978
	de	repair_de	Wartungskosten 1978
rep79	en	repair_en	Repair Record 1979
	nl	repair_nl	Onderhoud 1979
	de	repair_de	Wartungskosten 1979
foreign	en	origin_en	Car type
	nl	origin_nl	Auto type
	de	origin_de	Auto typos

```
Value labels by language-label pattern
pattern:   (en repair_en) (nl repair_nl) (de repair_de)
varlist:   rep78 rep79
repair_en:
           1 very bad
           2 bad
           3 reasonable
           4 good
           5 very good
repair_nl:
           1 zeer slecht
           2 slecht
           3 redelijk
           4 goed
           5 zeer goed
repair_de:
           1 sehr slecht
           2 slecht
           3 befriedigend
           4 gut
           5 sehr gut
pattern:   (en origin_en) (nl origin_nl) (de origin_de)
varlist:   foreign
```

```
origin_en:
           0 Domestic
           1 Foreign
origin_nl:
           0 Amerikaans
           1 Overig
origin_de:
           0 Amerikanisch
           1 sonstige
```

Focusing first on the `describe`-like table with the variable labels in all languages, you will notice that the variable labels for the variable `price` in Dutch (`nl`) and German (`de`) are the same. This was an honest mistake while I made the translation. This can be easily fixed:

```
. mlanguage select de : label var price "Preis"
```

Although in a more realistic application, I would change the file `adddе.do` rather than make an interactive change.

Second, `mlanguage list` has listed the value labels, grouped in "language-label patterns". This part of the output makes it easy to validate the collections of related value labels. Occasionally, this may also be useful if you find the texts of some value label too vague. If you are working with the English labels and tabulate the labels of `foreign`, you might find them vague. Foreign to what? Domestic where? You may find the labels in the other languages helpful in the interpretation.

```
. mlang list, var(foreign) nodes
Value labels by language-label pattern
pattern:   (de origin_de) (en origin_en) (nl origin_nl)
varlist:   foreign
origin_de:
           0 Amerikanisch
           1 sonstige
origin_en:
           0 Domestic
           1 Foreign
origin_nl:
           0 Amerikaans
           1 Overig
```

❑ Technical Note

Note that the order of the languages in the output corresponds to the ordering shown by `mlanguage dir`. In the first example of `mlanguage list`, we overruled the ordering to one we liked better. We could also have selected a subset of the available languages.

❑

5.4 Checking for integrity

After you have worked with a multilingual dataset for some time, the labeling information may need some maintenance work. You probably added variables that were variable or value labeled in the language in which you were working at that time, but you may have overlooked—or not have been able or willing—to add labels in the other languages as well. Maybe you have been working with colleagues who work in different languages, each adding variables with labels in different languages. Comparing the label sets across the languages "by hand" is possible, but is hardly amusing and certainly error prone in all but very small datasets. The subcommand `check` of `mlanguage` assists in the more complicated part of the job: verifying that the value labels are really defined and are consistent, complete, and compatible.

Consistency of value labels across languages means that if variables x_1 and x_2 share a value label in language L_1, they cannot be attached to different value labels in language L_2. For example, suppose that in `en`, the variables `rep78` and `rep79` share the value label `repair_en`, while in `nl`, `rep78` is value labeled by `repair_nl` and `rep79` by `origin_nl`. Then we would say that the value labeling is inconsistent. If the two variables share a value label, they must be in commensurable units, share meaning, etc. Such a relationship between the two variables holds in `en`, but not in `nl`. This difference in the relationships between variables is called "inconsistency". This form of inconsistency indicates a labeling problem that probably needs to be fixed. In the following example, I mixed up the value labels for repair record and for national origin of equipment. I should attach the correct value label to variable `rep79` in language `nl`. This need not be a serious problem, though. `check` only compares the names of the value labels, not the label contents. Absence of evidence (of commensurability) does not imply evidence of absence. You might attach two value labels that are in fact duplicates, i.e., consist of the same set of mapping. In this case, the problem is one of *redundancy*. Experts in database management stress that redundancy should be avoided because it threatens data integrity; the waste of resources is trivial in comparison. I recommend avoiding redundancy. In a companion article (Weesie 2005), I describe a command `labeldup` that identifies duplicate value labels and optionally eliminates them.

We say that the value labeling is *complete* if variables are either value labeled in all languages or in none. If `rep78` is value labeled in `en`, but not in `nl`, the labeling is incomplete. Incompleteness may be due to value labels, but also to variable labels and to the labels of the dataset. The subcommand `fix` facilitates completing the labeling through a procedure that resembles adding a new language.

Compatibility involves a comparison of the value labels that are linked to a variable in different languages. The value labels are compatible if they provide mappings for the same set of values in each of the languages. Suppose that the English value label `origin_en` of `foreign` consists of mappings for 0 and 1, say 0 → "Domestic" and 1 → "Foreign", while the German value label `origin_de` of `foreign` provides mappings for 0, 1, and 2, for example 0 → "Europa", 1 → "Amerika", and 2 → "Asien". Thus 2 is mapped in the German label but not in the English label. This will probably be an error that needs to be fixed.

To illustrate `mlanguage check`, I added variables `rep80`, `engine`, and `fuel`, which I tried to label, but I did not try very hard. In the English label set, we have

```
. mlang select en
. des
Contains data from cardata.dta
  obs:            74                          1978 Automobile Data
 vars:             8                          18 Apr 2005 16:32
 size:         2,146 (99.9% of memory free)   (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str17  %-17s                  Make and Model
price           int    %8.0gc                 Price
rep78           byte   %13.0g      repair_en
                                               Repair Record 1978
rep79           byte   %12.0g      repair_en
                                               Repair Record 1979
foreign         byte   %12.0g      origin_en
                                               Car type
rep80           byte   %11.0g                 Repair Record 1980
engine          byte   %12.0g      origin_en
                                               Engine type
fuel            byte   %13.0g      fuel_en    Type of fuel
-------------------------------------------------------------------------------
Sorted by:  foreign
     Note:  dataset has changed since last saved
```

How well did I do?

```
. mlanguage check
Inconsistency in value labeling found
Check  variable: rep80
             de: repair_nl
             en:
             nl: repair_nl
Against varlist: rep78 rep79
             de: repair_de
             en: repair_en
             nl: repair_nl
r(198);
```

I made a mistake, attaching to `rep80` in German the Dutch label `repair_nl` (inconsistency), and I forgot to attach a value label in English (incompleteness). Inconsistency has to be solved manually:

```
. mlang select de : label value rep80 repair_de
```

Incompleteness may be resolved by hand

```
. mlang select en : label value rep80 repair_en
```

but you may want the assistance of `mlanguage fix`. After making these changes, also make sure that `mlanguage check` does not find incompatibilities by specifying the option `same`.

```
. mlanguage check, same
value labeling is consistent
value label fuel_nl not yet defined
additional value labels have to be defined
value labels are compatible
```

Note that `mlanguage check` has also detected that the variable `fuel` is value labeled in Dutch by the label `fuel_nl` but that this label has not yet been defined. There is work to be done!

5.5 Fixing incompleteness

`mlanguage fix` helps make the labeling information *complete*. If some element of the data (dataset, variables, categories) is labeled in one language, it should be labeled in all languages. The command does *not* demand that everything always be labeled. `mlanguage fix` is perfectly happy if in a dataset with labels in `en`, `nl`, and `de`, the variable `zipcode` has no variable or value labels in any of the languages. However, if the variable `rep78` is variable labeled in `en`, it assumes that you also want to add variable labels for `rep78` in `nl` and `de` and creates corresponding `label variable` statements. This sounds easy for variable labels and for the data label, but what about for value labels? There are really two possibilities. You may already have defined the value label in, say, `nl` to be the value label of `rep80` (e.g., you used it to value label `rep78`) and failed to make the link. Alternatively, you may need to define new value labels by translating the English value labels. `mlanguage fix` tries to make a reasonable guess on what you want by exploiting the requirement that value labels be *consistent* across languages in the sense explained in the previous subsection. `mlanguage fix` thus will fail if the value labeling is currently inconsistent. How consistency helps `mlanguage fix` make an educated guess is easily explained by an example. Suppose that in `en`, the variables `rep78` and `rep79` share the value label `repair_en`. Furthermore, in `nl`, the variable `rep78` is value labeled by `repair_nl`, but `rep79` is not value labeled. Finally, in `de`, neither `rep78` nor `rep79` are value labeled. Now `mlanguage fix` works as follows. The variables `rep78` and `rep79` have a common value label in one language (in this example `en`); hence, they are "of the same type". This is a conceptual relation between variables, and it should hold, irrespective of language. Thus we can conclude that the variables should also share a value label in `nl` and in `de`. In the language `nl`, this is simple. The value label for `rep79` must be the one that is already attached to `rep78`; thus `mlanguage fix` writes in *todo*`_nl.do` one statement to fix the missing link:

```
label value rep78 repair_nl
```

In the language `de`, the situation is different. `fix` has inferred that the `repair` variables must be value labeled because they are in `en`. However, we do not already have a value label for this in `de`; at least, `mlanguage fix` does not know of any. Thus `mlanguage fix` must produce code to define the value label in `de`, named `repair_de`, and to attach this new label to the variables:

```
label define repair_de 1 "very bad"
label define repair_de 2 "bad"
...
label value rep78 repair_de
label value rep79 repair_de
```

Similar to the `mlanguage add` subcommand, `mlanguage fix` creates a series of files, one for each language, with the label information that you must supply. These files are very similar to the do-files produced by `mlanguage add` but include only the labels needed to make the labeling complete. You must translate (or have someone else do it) the labels in each of the do-files and execute the do-files, and voila, the dataset is in good shape again.

A careful reader may now wonder about the translation. In the case of `mlanguage add`, the labels were initialized to the values in the current language, so all translations are from the current language to the added language. With `mlanguage fix`, the case is more complicated if you are dealing with more than two languages. There are two issues here, which we will discuss in the context of our three-language example. Suppose that a variable label is missing in German. If the corresponding variable label is available in only one of the other languages, that label must be used as an initial value for translation. But what if both English and Dutch versions are available? `mlanguage fix` must choose which one to use. For this purpose, `mlanguage fix` uses an ordering of the languages; the first language in this ordering in which the label is available is used for initialization. The ordering is just the *languagenamelist* argument to `mlanguage fix`; if this list is not specified, `mlanguage fix` uses the ordering of the available languages as reported by `mlanguage dir`.

Second, the fix-file for the German label set may contain some labels initialized in English and other labels in Dutch. We include the name of the "source language" in the comments so that different people can work on "their" translations.[1] Enough words—let's see some action.

```
. mlanguage fix, saving(F) replace noinstruct
Language de needs additional labeling; see file F_de.do
Language en needs additional labeling; see file F_en.do
Language nl needs additional labeling; see file F_nl.do
```

Since I planned to display the files, I suppressed the instructions. The file `F_en.do` consists of the labeling needed to make the English label set complete.

```
. type F_en.do
label language en
// the rest of this file attaches value labels to variables
// you need not make any changes below this point
label value rep80 repair_en
```

[1] If this design, in which labels are grouped by target language, is inconvenient, you may split the files using an editor or a `grep`-like utility. If this does not work well, please contact me, and I may consider adding an option to split the labels in (source, target) specific files.

The first line activates English (`en`). We are instructed that the value label `repair_en` must be attached to `rep80`, but we need not do anything for this—apart from running the file `F_en.do`.

Similarly, we look at the files `F_nl.do` and `F_de.do`:

```
. type F_nl.do
label language nl

label define fuel_nl 1 `"gas"'                          // en: gas
label define fuel_nl 2 `"diesel"' , add                 // en: diesel
label define fuel_nl 3 `"liquid gas"' , add             // en: liquid gas
label define fuel_nl 4 `"vegetable oil"' , add          // en: vegetable oil

// the rest of this file attaches value labels to variables
// you need not make any changes below this point


. type F_de.do
label language de

label variable rep80 `"Repair Record 1980"'             // en: Repair Record 1980
label variable fuel `"Type of fuel"'                    // en: Type of fuel

label define fuel_de 1 `"gas"'                          // en: gas
label define fuel_de 2 `"diesel"' , add                 // en: diesel
label define fuel_de 3 `"liquid gas"' , add             // en: liquid gas
label define fuel_de 4 `"vegetable oil"' , add          // en: vegetable oil

// the rest of this file attaches value labels to variables
// you need not make any changes below this point

label value fuel fuel_de
```

In `F_nl.do`, we see that statements are included to define the Dutch version of value label for `fuel`. No statement to attach the label is included—the link is already established. The German label requires the most work: Two variable labels and one value label must be translated. If the translations are done, all we have to do is to run the scripts, and the labeling is complete again.

❑ Technical Note

If the labeling of a language is consistent and complete, adding a new language *ln* with

```
. language add ln, saving(toln)
```

is equivalent to adding an empty language and then fixing the problem!

```
. label language ln, new
. language fix, saving(toln)
```

❑

6 References

Weesie, J. 2005. Value label utilities: labeldup and labelrename. *Stata Journal* 5(2): 14–21.

About the Author

Jeroen Weesie is associate professor of Mathematical Sociology at the Department of Sociology at Utrecht University.