The Stata Journal (2007)
7, Number 4, pp. 582–583

Stata tip 52: Generating composite categorical variables

Nicholas J. Cox
Department of Geography
Durham University
Durham City, UK
n.j.cox@durham.ac.uk

If you have two or more categorical variables, you may want to create one composite categorical variable that can take on all the possible joint values. The canonical example for Stata users is given by cross-combinations of `foreign` and `rep78` in the `auto` data. Setting aside missings, `foreign` takes on values of 0 and 1, and `rep78` takes on values of 1, 2, 3, 4, and 5. Hence there are ten possible joint values, which could be 0 and 1, 0 and 2, and so forth. As it happens, only eight occur in the data. If we add the value labels attached to `foreign`, we have Domestic 1, Domestic 2, and so forth.

Writing the values like that raises the question of whether these cross-combinations will be better expressed as string variables or as numeric variables with value labels. On the whole, an integer-valued numeric variable with value labels defined and attached is the best arrangement for any categorical variable, but a string variable may also be convenient, especially if you are producing a kind of composite identifier.

A method often seen is to produce string variables with `tostring` (see [D] **destring**), for example,

```
. tostring foreign rep78, generate(Foreign Rep78)
. gen both = Foreign + Rep78
```

Naturally, there are endless minor variations on this method. A small but useful improvement is to insert a space or other punctuation:

```
. gen both = Foreign + " " + Rep78
```

However, this method is not especially good. `tostring` is really for correcting mistakes, whether attributable to human fault or to some software used before you entered Stata: some variable that should be string is in fact numeric. You need to correct that mistake. `tostring` is a safe way of doing that.

That intended purpose does not stop `tostring` being useful for things for which it was not intended, but there are two specific disadvantages to this method:

1. This method needs two lines, and you can do it in one. That is a little deal.
2. This method could lose information, especially for variables with value labels or with noninteger values. That is, potentially, a big deal.

The second point may suggest using `decode` instead, but my suggestions differ. A better method is to use `egen, group()`. See [D] **egen**.


```
. egen both = group(foreign rep78), label
```

This command produces a new numeric variable, with integer values 1 and above, and value labels defined and attached. Particularly, note the `label` option, which is frequently overlooked.

This method has several advantages:

1. One line.
2. No loss of information. Observations that are identical on the arguments are identical on the results. Value labels are used, not ignored. Distinct noninteger values will also remain distinct.
3. The label is useful—indeed essential—for tables and graphs to make sense.
4. Efficient storage.
5. Extends readily to three or more variables.

Another fairly good method is to use `egen, concat()`.

```
. egen both = concat(foreign rep78), decode p(" ")
```

This command creates a string variable, so it is less efficient for data storage and is less versatile for graphics or modeling. Compared with `tostring`, the advantages are

1. One line.
2. You can mix numeric and string arguments. `concat()` will calculate what is needed.
3. You can use the `decode` option to use value labels on the fly.
4. You can specify punctuation as separator, here a blank.
5. Extends to three or more variables.