char — Characteristics

Description

See [U] **12.8 Characteristics** for a description of characteristics. These commands allow manipulating characteristics.

Syntax

Define characteristics

    char [define] *evarname*[*charname*] [["]*text*["]]

List characteristics

    char list [*evarname*[[*charname*]]]

Rename characteristics

    char rename *oldvar newvar* [, replace]

Also related is

    {local | global} *mname*: char *evarname*[[*charname*]]

evarname is a variable name or _dta and *charname* is a characteristic name. In the syntax diagrams, distinguish carefully between [], which you type, and [], which indicates that the element is optional.

Option

replace (for use only with char rename) specifies that if characteristics of the same name already exist, they are to be replaced. replace is a seldom-used, low-level, programmer's option.

char rename *oldvar newvar* moves all characteristics of *oldvar* to *newvar*, leaving *oldvar* with none and *newvar* with all the characteristics *oldvar* previously had. char rename *oldvar newvar* moves the characteristics, but only if *newvar* has no characteristics with the same name. Otherwise, char rename produces the error message that *newvar*[*whatever*] already exists.

Remarks and examples stata.com

We begin by showing how the commands work mechanically and then continue to demonstrate the commands in more realistic situations.

`char define` sets and clears characteristics, although there is no reason to type `define`:

```
. char _dta[one] this is char named one of _dta
. char _dta[two] this is char named two of _dta
. char mpg[one]  this is char named   one    of mpg
. char mpg[two] "this is char named   two    of mpg"
. char mpg[three] "this is char named three of mpg"
```

Whether we include the double quotes does not matter. You clear a characteristic by defining it to be nothing:

```
. char mpg[three]
```

`char list` is used to list existing characteristics; it is typically used for debugging:

```
. char list
      _dta[two]      :  this is char named two of _dta
      _dta[one]      :  this is char named one of _dta
       mpg[two]      :  this is char named   two    of mpg
       mpg[one]      :  this is char named   one    of mpg
. char list _dta[]
      _dta[two]      :  this is char named two of _dta
      _dta[one]      :  this is char named one of _dta
. char list mpg[]
       mpg[two]      :  this is char named   two    of mpg
       mpg[one]      :  this is char named   one    of mpg
. char list mpg[one]
       mpg[one]      :  this is char named   one    of mpg
```

The order may surprise you—it is the way it is because of how Stata's memory-management routines work—but it does not matter.

`char rename` moves all the characteristics associated with *oldvar* to *newvar*:

```
. char rename mpg weight
. char list
      _dta[two]      :  this is char named two of _dta
      _dta[one]      :  this is char named one of _dta
    weight[two]      :  this is char named   two    of mpg
    weight[one]      :  this is char named   one    of mpg
. char rename weight mpg                      // put it back
```

The contents of specific characteristics may be obtained in the same way as local macros by referring to the characteristic name between left and right single quotes; see [U] **12.8 Characteristics**.

```
. display "`mpg[one]'"
this is char named   one    of mpg
. display "`_dta[]'"
two one
```

Referring to a nonexisting characteristic returns a null string:

```
. display "the value is |`mpg[three]'|"
the value is ||
```

How to program with characteristics

▷ Example 1

You are writing a program that requires the value of the variable recording "instance" (first time, second time, etc.). You want your command to have an option ins(*varname*), but after the user has specified the variable once, you want your program to remember it in the future, even across sessions. An outline of your program is

```
program ...
        version 15.0
        syntax ... [, ... ins(varname) ... ]
        ...
        if "`ins'"=="" {
                local ins "`_dta[Instance]'"
        }
        confirm variable `ins'
        char _dta[Instance] : `ins'
        ...
end
```

◁

▷ Example 2

You write a program, and among other things, it changes the contents of one of the variables in the user's data. You worry about the user pressing *Break* while the program is in the midst of the change, so you correctly decide to construct the replaced values in a temporary variable and, only at the conclusion, drop the user's original variable and replace it with the new one. In this example, macro `uservar' contains the name of the user's original variable. Macro `newvar' contains the name of the temporary variable that will ultimately replace it.

The following issues arise when you duplicate the original variable: you want the new variable to have the same variable label, the same value label, the same format, and the same characteristics.

```
program ...
        version 15.0
        ...
        tempvar newvar
        ...
        ( code creating `newvar')
        ...
        local varlab : variable label `uservar'
        local vallab : value label `uservar'
        local format : format `uservar'
        label var `newvar' "`varlab'"
        label values `newvar' `vallab'
        format `newvar' `format'
        char rename `uservar' `newvar'
        drop `uservar'
        rename `newvar' `uservar'
end
```

You are supposed to notice the char rename command included to move the characteristics originally attached to `uservar' to `newvar'. See [P] **macro**, [D] **label**, and [D] **format** for information on the commands preceding the char rename command.

    This code is almost perfect, but if you are really concerned about the user pressing *Break*, there
is a potential problem. What happens if the user presses *Break* between the char rename and the
final rename? The last three lines would be better written as

```
nobreak {
        char rename 'uservar' 'newvar'
        drop 'uservar'
        rename 'newvar' 'uservar'
}
```

Now even if the user presses *Break* during these last three lines, it will be ignored; see [P] **break**.

◁

Also see

[P] **macro** — Macro definition and manipulation

[D] **notes** — Place notes in data

[U] **12.8 Characteristics**

[U] **18.3.6 Extended macro functions**

[U] **18.3.13 Referring to characteristics**