Title

stata.com

macro lists — Manipulate lists

Description Syntax Remarks and examples Also see

Description

The macro function `list` manipulates lists. See [P] **macro** for other macro functions.

`uniq` A returns A with duplicate elements removed. The resulting list has the same ordering of its elements as A; duplicate elements are removed from their rightmost position. If $A =$ "$a\ b\ a\ c\ a$", `uniq` returns "$a\ b\ c$".

`dups` A returns the duplicate elements of A. If $A =$ "$a\ b\ a\ c\ a$", `dups` returns "$a\ a$".

`sort` A returns A with its elements placed in alphabetical (ascending ASCII or code-point) order.

`rsort` A returns A with its elements placed in reverse alphabetical (descending ASCII or code-point) order.

`retokenize` A returns A with single spaces between elements. Logically speaking, it makes no difference how many spaces a list has between elements, and thus `retokenize` leaves the list logically unchanged.

`clean` A returns A retokenized and with each element adorned minimally. An element is said to be unadorned if it is not enclosed in quotes (for example, a). An element may also be adorned in simple or compound quotes (for example, `"`a`"` or `‘"`a`"’`). Logically speaking, it makes no difference how elements are adorned, assuming that they are adorned adequately. The list

`‘"`a`"’ ‘"`$b\ c$`"’ ‘"`b `"`c`" `d`"’`

is equal to

a `"`$b\ c$`" ‘"`b `"`c`" `d`"’`

`clean`, in addition to performing the actions of `retokenize`, adorns each element minimally: not at all if the element contains no spaces or quotes, in simple quotes (`"` and `"`) if it contains spaces but not quotes, and in compound quotes (`‘"` and `"’`) otherwise.

A `|` B returns the union of A and B, the result being equal to A with elements of B not found in A added to the tail. For instance, if $A =$ "$a\ b\ c$" and $B =$ "$b\ d\ e$", A `|` B is "$a\ b\ c\ d\ e$". If you instead want list concatenation, you code,

`local` *newlist* `‘"‘`A`’ ‘`B`’"’`

In the example above, this would return "$a\ b\ c\ b\ d\ e$".

A `&` B returns the intersection of A and B. If $A =$ "$a\ b\ c\ d$" and $B =$ "$b\ c\ f\ g$", then A `&` $B =$ "$b\ c$".

A `-` B returns a list containing elements of A with the elements of B removed, with the resulting elements in the same order as A. For instance, if $A =$ "$a\ b\ c\ d$" and $B =$ "$b\ e$", the result is "$a\ c\ d$".

A `==` B returns 0 or 1; it returns 1 if A is equal to B, that is, if A has the same elements as B and in the same order. Otherwise, 0 is returned.

A `===` *B* returns 0 or 1; it returns 1 if *A* is equivalent to *B*, that is, if *A* has the same elements as *B* regardless of the order in which the elements appear. Otherwise, 0 is returned.

A `in` *B* returns 0 or 1; it returns 1 if all elements of *A* are found in *B*. If *A* is empty, `in` returns 1. Otherwise, 0 is returned.

`sizeof` *A* returns the number of elements of *A*. If *A* = "*a b c*", `sizeof` *A* is 3. (`sizeof` returns the same result as the macro function `word count`; see *Macro functions for parsing* under *Syntax* in [P] **macro**.)

`posof "`*element*`" in` *A* returns the location of *macname* in *A* or returns 0 if not found. For instance, if *A* contains "*a b c d*", then `posof "`*b*`" in` *A* returns 2. (`word # of` may be used to extract positional elements from lists, as can `tokenize` and `gettoken`; see *Macro functions for parsing* under *Syntax* in [P] **macro** and also see [P] **tokenize** and [P] **gettoken**.)

It is the element itself and not a macroname that you type as the first argument. In a program where macro `tofind` contained an element to be found in list (macro) `variables`, you might code

```
local i : list posof `""`tofind'""' in variables
```

element must be enclosed in simple or compound quotes.

Syntax

{`local`|`global`} *macname* `:` `list uniq` *macname*

{`local`|`global`} *macname* `:` `list dups` *macname*

{`local`|`global`} *macname* `:` `list sort` *macname*

{`local`|`global`} *macname* `:` `list rsort` *macname*

{`local`|`global`} *macname* `:` `list retokenize` *macname*

{`local`|`global`} *macname* `:` `list clean` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `|` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `&` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `-` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `==` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `===` *macname*

{`local`|`global`} *macname* `:` `list` *macname* `in` *macname*

{`local`|`global`} *macname* `:` `list sizeof` *macname*

{`local`|`global`} *macname* `:` `list posof "`*element*`" in` *macname*

Note: Where *macname* appears above, it is the name of a macro and *not* its contents that you are to type. For example, you are to type

```
local result : list list1 | list2
```

and not

```
local result : list "‘list1’" | "‘list2’"
```

*macname*s that appear to the right of the colon are assumed to be the names of local macros. You may type `local(`*macname*`)` to emphasize that fact. Type `global(`*macname*`)` if you wish to refer to a global macro.

Remarks and examples

stata.com

Remarks are presented under the following headings:

A *list* is a space-separated set of elements listed one after the other. The individual elements may be enclosed in quotes, and elements containing spaces obviously must be enclosed in quotes. The following are examples of lists:

```
this that what
"first element" second "third element" 4
this that what this that
```

Also a list could be empty.

Do not confuse varlist with list. Varlists are a special notation, such as `"id m* pop*"`, which is a shorthand way of specifying a list of variables; see **[U] 11.4 varname and varlists**. Once expanded, however, a varlist is a list.

Treatment of adornment

An element of a list is said to be adorned if it is enclosed in quotes. Adornment, however, plays no role in the substantive interpretation of lists. The list

a `"`b`"` c

is identical to the list

$a\ b\ c$

Treatment of duplicate elements in lists

With the exception of `uniq` and `dups`, all list functions treat duplicates as being distinct. For instance, consider the list A,

$$a\ b\ c\ b$$

Notice that b appears twice in this list. You want to think of the list as containing a, the first occurrence of b, c, and the second occurrence of b:

$$a\ b_1\ c\ b_2$$

Do the same thing with the duplicate elements of all lists, carry out the operation on the now unique elements, and then erase the subscripts from the result.

If you were to ask whether B = "b b" is `in` A, the answer would be yes, because A contains two occurrences of b. If B contained "b b b", however, the answer would be no because A does not contain three occurrences of b.

Similarly, if B = "b b", then A | B = "a b c b", but if B = "b b b", then A | B = "a b c b b".

Also see

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