confirm — Argument verification

Description Syntax Option Remarks and examples Also see

Description

`confirm` verifies that the arguments following `confirm` ... are of the claimed type and issues the appropriate error message and nonzero return code if they are not.

`confirm` is useful in do-files and programs when you do not want to bother issuing your own error message. `confirm` can also be combined with `capture` to detect and handle error conditions before they arise; see [P] **capture**.

Syntax

`confirm existence` *string*

`confirm [new] file` *filename*

`confirm [numeric | string | date] format` *string*

`confirm [new] frame` *name*

`confirm names` *names*

`confirm [integer] number` *string*

`confirm matrix` *string*

`confirm scalar` *string*

`confirm [new | numeric | string | str# | alias |` *type*`] variable` *varlist* `[, exact]`

where *type* is {`byte` | `int` | `long` | `float` | `double` | `str#` | `strL`}

Option

`exact` specifies that a match be declared only if the names specified in *varlist* match. By default, names that are abbreviations of variables are considered to be a match.

Remarks and examples

Remarks are presented under the following headings:

confirm existence

`confirm existence` displays the message " " found where something expected" and produces a return code of 6 if *string* does not exist.

confirm file

`confirm file` verifies that *filename* exists and is readable and issues the appropriate error message and return code if not.

`confirm new file` verifies that *filename* does not exist and that *filename* could be opened for writing, and issues the appropriate error message and return code if not.

The possible error messages and return codes are

Message	Return code
`____ found where filename expected`	7
`file ____ not found`	601
`file ____ already exists`	602
`file ____ could not be opened`	603

Return codes of 7 and 603 are possible for both `confirm file` and `confirm new file`. For `confirm new file`, a return code of 603 indicates that the filename is invalid, the specified directory does not exist, or the directory permissions do not allow you to create a new file. For instance, even if *filename* does not exist, `confirm new file` *newdir*`\`*newfile* will generate an error if *newdir* does not exist and if you do not have permissions to create a file in *newdir*. `confirm new file` *filename* will fail if you do not have adequate permissions to create a new file in the current working directory.

confirm format

`confirm format` verifies that *string* is a valid variable display format. It produces the message

'string' `found where format expected`

with a return code of 7 if the format is not valid. It produces the message

`" found where format expected`

with a return code of 7 if the format is empty.

`confirm numeric format` specifies that the argument must be a valid numeric format. Valid numeric formats are general, fixed, and exponential. If not, it produces a return code of 7 and the message

'string' `found where numeric format expected`

or

`" found where numeric format expected`

if *string* is empty.

`confirm string format` specifies that the argument must be a valid string format. If not, it produces a return code of 7 and the message

'string' found where string format expected

or

" found where string format expected

if *string* is empty.

`confirm date format` specifies that the argument must be a valid date format. If not, it produces a return code of 7 and the message

'string' found where date format expected

or

" found where date format expected

if *string* is empty.

confirm frame

`confirm frame` verifies that *name* is a frame (see [D] **frames**). It produces the message

frame *name* not found

with a return code of 111 if a frame named *name* does not exist.

`confirm new frame` verifies that *name* is valid to be used as the name of a frame and that a frame with that name does not already exist. The possible messages and return codes are the following:

Message	Return code
____ found where frame name expected	7
frame ____ already defined	110
____ invalid name	198

confirm names

`confirm names` verifies that the argument or arguments are valid names according to Stata's naming conventions. It produces the message

{*name*|nothing} invalid name

with a return code of 7 if the names are not valid.

confirm number

`confirm number` verifies that the argument can be interpreted as a number, such as `1`, `5.2`, `-5.2`, or `2.5e+10`. It produces the message

{*string*|`nothing`} `found where number expected`

with a return code of 7 if not.

`confirm integer number` specifies that the argument must be an integer, such as 1 or 2.5e+10, but not 5.2 or −5.2. If not, it produces a return code of 7 and a slight variation on the message above:

{*string*|`nothing`} `found where integer expected`

confirm matrix

`confirm matrix` verifies that *string* is a matrix. It produces the message

`matrix` *string* `not found`

with a return code of 111 if *string* is not a matrix.

confirm scalar

`confirm scalar` verifies that *string* is a scalar. It produces the message

`scalar` *string* `not found`

with a return code of 111 if *string* is not a scalar.

confirm variable

`confirm variable` verifies that *varlist* can be interpreted as an existing varlist of any types of variables. If not, the appropriate error message and nonzero return code are returned:

Message	Return code
`____ found where numeric variable expected`	7
`____ found where string variable expected`	7
`____ found where str# variable expected`	7
`____ found where strL variable expected`	7
`____ found where alias variable expected`	7
`no variables defined`	111
`variable ____ not found`	111
`____ invalid name`	198

`confirm numeric variable` specifies that all the variables are numeric. If the variable exists but is not numeric, Stata displays the message

'varname' `found where numeric variable expected`

or

`" found where numeric variable expected`

with a return code of 7 if *varlist* is not specified.

`confirm string variable` specifies that all the variables are strings, meaning `str#` or `strL`. If the variable exists but is not a string variable, Stata displays the message

```
'varname' found where string variable expected
```

or

```
" found where string variable expected
```

with a return code of 7 if *varlist* is not specified.

`confirm str# variable` specifies that all the variables are `str#`, such as `str10` or `str42`, but are not `strL`s.

`confirm alias variable` specifies that all the variables were created by `fralias add`. If the variable exists but was not created by `fralias add`, Stata displays the message

```
'varname' found where alias variable expected
```

`confirm` *type* `variable` specifies that all variables are of the indicated storage type. For example, `confirm int variable myvar`, `confirm float variable myvar thatvar`, or `confirm strL variable blobvar`. As with `confirm string variable`, the appropriate message and return code of 7 are possible. When there is an alias variable in *varlist*, the linked variable's storage type is checked.

`confirm new variable` verifies that *varlist* can be interpreted as a new varlist. The possible messages and return codes are

Message	Return code
`____ found where varname expected`	7
`____ already defined`	110
`____ invalid name`	198

Example 1

`confirm` is a cheap way to include minimal syntax checking in your programs. For instance, you have written a program that is supposed to take a one-integer argument. Although you do not have to include any syntax checking at all—the program will probably fail with some error if the argument is incorrect—it is safer to add one line at the top of the program:

```
confirm integer number `1'
```

Now if the first argument is not an integer, you will get a reasonable error message, and the program will stop automatically.

▷ Example 2

More sophisticated programs often combine the `confirm` and `capture` commands. For instance, `ttest` has a complex syntax: if the user types `ttest var=5`, it tests that the mean of `var` is 5 using one set of formulas, and if the user types `ttest var=var2`, it tests equality of means by using another set of formulas. Whether there is a number or a variable to the right of the equal sign determines which set of formulas `ttest` uses. This choice was done by

```
capture confirm number `exp'
if _rc==0 {
        (code for test against a constant)
        exit
}
(code for test of two variables)
```

◁

Also see

[P] **capture** — Capture return code

[D] **fralias** — Alias variables from linked frames