Title

stata.com

st_addvar( ) — Add variable to current Stata dataset

Syntax Description Remarks and examples Conformability
Diagnostics Reference Also see

Syntax

real rowvector `st_addvar(`*type*`,` *name*`)`

real rowvector `st_addvar(`*type*`,` *name*`,` *nofill*`)`

real rowvector `_st_addvar(`*type*`,` *name*`)`

real rowvector `_st_addvar(`*type*`,` *name*`,` *nofill*`)`

where

type: *string scalar* or *rowvector* containing `"byte"`, `"int"`, `"long"`, `"float"`, `"double"`, `"str#"`, or `"strL"`
or
real scalar or *rowvector* containing # (interpreted as `str#`)

name: *string rowvector* containing new variable names

nofill: *real scalar* containing 0 or non-0

Description

`st_addvar(`*type*`,` *name*`)` adds new variable *name*(s) of type *type* to the Stata dataset. Returned are the variable indices of the new variables. `st_addvar()` aborts with error (and adds no variables) if any of the variables already exist or cannot be added for other reasons.

`st_addvar(`*type*`,` *name*`,` *nofill*`)` does the same thing. *nofill* $\neq 0$ specifies that the variables' values are not to be filled in with missing values. `st_addvar(`*type*`,` *name*`, 0)` is the same as `st_addvar(`*type*`,` *name*`)`. Use of *nofill* $\neq 0$ is not, in general, recommended. See *Using nofill* in *Remarks and examples* below.

`_st_addvar()` does the same thing as `st_addvar()` except that, rather than aborting with error if the new variable cannot be added, returned is a 1×1 scalar containing the negative of the appropriate Stata return code.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Creating a new variable
Creating new variables
Creating new string variables
Creating a new temporary variable
Creating temporary variables
Handling errors
Using nofill

Creating a new variable

To create new variable myvar as a double, code

```
idx = st_addvar("double", "myvar")
```

or

```
(void) st_addvar("double", "myvar")
```

You use the first form if you will subsequently need the variable's index number, or you use the second form otherwise.

Creating new variables

You can add more than one variable. For instance,

```
idx = st_addvar("double", ("myvar1","myvar2"))
```

adds two new variables, both of type double.

```
idx = st_addvar(("double","float"), ("myvar1","myvar2"))
```

also adds two new variables, but this time, myvar1 is double and myvar2 is float.

Creating new string variables

Creating string variables is no different from any other type:

```
idx = st_addvar(("str10","str5"), ("myvar1","myvar2"))
```

creates myvar1 as a str10 and myvar2 as a str5.

There is, however, another way to specify the types.

```
idx = st_addvar((10,5), ("myvar1","myvar2"))
```

also creates myvar1 as a str10 and myvar2 as a str5.

```
idx = st_addvar(10, ("myvar1","myvar2"))
```

creates both variables as str10s.

Creating a new temporary variable

Function st_tempname() (see [M-5] **st_tempname( )**) returns temporary variable names. To create a temporary variable as a double, code

```
idx = st_addvar("double", st_tempname())
```

or code

```
(void) st_addvar("double", name=st_tempname())
```

You use the first form if you will subsequently need the variable's index, or you use the second form if you will subsequently need the variable's name. You will certainly need one or the other. If you will need both, code

```
idx = st_addvar("double", name=st_tempname())
```

Creating temporary variables

`st_tempname()` can return a vector of temporary variable names.

```
idx = st_addvar("double", st_tempname(5))
```

creates five temporary variables, each of type `double`.

Handling errors

There are three common reasons why `st_addvar()` might fail: the variable name is invalid or a variable under that name already exists or there is insufficient memory to add another variable. If there is a problem adding a variable, `st_addvar()` will abort with error. If you wish to avoid the traceback log and just have Stata issue an error, use `_st_addvar()` and code

```
if ((idx = _st_addvar("double", "myvar"))<0) exit(error(-idx))
```

If you are adding multiple variables, look at the first element of what `_st_addvar()` returns:

```
if ((idx = _st_addvar(types, names))[1]<0) exit(error(-idx))
```

Using nofill

The three-argument versions of `st_addvar()` and `_st_addvar()` allow you to avoid filling in the values of the newly created variable. Filling in those values with missing really is a waste of time if the next thing you are going to do is fill in the values with something else. On the other hand, it is important that all the observations be filled in on the new variable before control is returned to Stata, and this includes returning to Stata because of subsequent error or the user pressing *Break*. Thus use of *nofill* $\neq 0$ is not, in general, recommended. Filling in values really does not take that long.

If you are determined to save the computer time, however, see [M-5] **setbreakintr( )**. To do things right, you need to set the break key off, create your variable, fill it in, and turn break-key processing back on.

There is, however, a case in which use of *nofill* $\neq 0$ is acceptable and such effort is not required: when you are creating a temporary variable. Temporary variables vanish in any case, and it does not matter whether they are filled in before they vanish.

Temporary variables in fact vanish not when Mata ends but when the ado-file calling Mata ends, if there is an ado-file. We will assume there is an ado-file because that is the only case in which you would be creating a temporary variable anyway. Because they do not disappear until later, there is the possibility of there being an issue if the variable is not filled in. If we assume, however, that your Mata program is correctly written and does fill in the variable ultimately, then the chances of a problem are minimal. If the user presses *Break* or there is some other problem in your program that causes Mata to abort, the ado-file will be aborted, too, and the variable will vanish.

Let us add that Stata will not crash if a variable is not filled in, even if it regains control. The danger is that the user will look at the variable or, worse, use it and be baffled by what he or she sees, which might concern not only odd values but also NaNs and worse.

Conformability

`st_addvar(`*type*`,` *name*`,` *nofill*`)`:

type:	1×1	or	$1 \times k$
name:	$1 \times k$		
nofill:	1×1	(optional)	
result:	$1 \times k$		

`_st_addvar(`*type*`,` *name*`,` *nofill*`)`:

type:	1×1	or	$1 \times k$
name:	$1 \times k$		
nofill:	1×1	(optional)	
result:	$1 \times k$	or, if error,	1×1

Diagnostics

`st_addvar(`*type*`,` *name*`,` *nofill*`)` aborts with error if

1. *type* is not equal to a valid Stata variable type and it is not a number that would form a valid `str#` variable type;
2. *name* is not a valid variable name;
3. a variable named *name* already exists;
4. there is insufficient memory to add another variable.

`_st_addvar(`*type*`,` *name*`,` *nofill*`)` aborts with error for reason 1 above, but otherwise, it returns the negative value of the appropriate Stata return code.

Both functions, when creating multiple variables, create either all the variables or none of them. Whether creating one variable or many, if variables are created, `st_updata()` (see [M-5] **st_updata( )**) is set unless all variables are temporary; see [M-5] **st_tempname( )**.

Reference

Gould, W. W. 2006. Mata Matters: Creating new variables—sounds boring, isn't. *Stata Journal* 6: 112–123.

Also see

[M-5] **st_store( )** — Modify values stored in current Stata dataset

[M-5] **st_tempname( )** — Temporary Stata names

[M-4] **stata** — Stata interface functions