capture — Capture return code

Syntax

capture [ : ] command

capture {
    stata_commands
}

Description

capture executes command, suppressing all its output (including error messages, if any) and issuing a return code of zero. The actual return code generated by command is stored in the built-in scalar _rc.

capture can be combined with {} to produce capture blocks, which suppress output for the block of commands. See the technical note following example 6 for more information.

Remarks and examples

capture is useful in do-files and programs because their execution terminates when a command issues a nonzero return code. Preceding sensitive commands with the word capture allows the do-file or program to continue despite errors. Also do-files and programs can be made to respond appropriately to any situation by conditioning their remaining actions on the contents of the scalar _rc.

Example 1

You will never have cause to use capture interactively, but an interactive experiment will demonstrate what capture does:

```
    . drop _all
    . list myvar
    no variables defined
    r(111);
    . capture list myvar
    . display _rc
    111
```

When we said list myvar, we were told that we had no variables defined and got a return code of 111. When we said capture list myvar, we got no output and a zero return code. First, you should wonder what happened to the message “no variables defined”. capture suppressed that message. It suppresses all output produced by the command it is capturing. Next we see no return code message, so the return code was zero. We already know that typing list myvar generates a return code of 111, so capture suppressed that, too.
capture places the return code in the built-in scalar _rc. When we display the value of this scalar, we see that it is 111.

Example 2

Now that we know what capture does, let’s put it to use. capture is used in programs and do-files. Sometimes you will write programs that do not care about the outcome of a Stata command. You may want to ensure, for instance, that some variable does not exist in the dataset. You could do so by including capture drop result.

If result exists, it is now gone. If it did not exist, drop did nothing, and its nonzero return code and the error message have been intercepted. The program (or do-file) continues in any case. If you have written a program that creates a variable named result, it would be good practice to begin such a program with capture drop result. This way, you could use the program repeatedly without having to worry whether the result variable already exists.

Technical note

When combining capture and drop, never say something like capture drop var1 var2 var3. Remember that Stata commands do either exactly what you say or nothing at all. We might think that our command would be guaranteed to eliminate var1, var2, and var3 from the data if they exist. It is not. Imagine that var3 did not exist in the data. drop would then do nothing. It would not drop var1 and var2. To achieve the desired result, we must give three commands:

```
capture drop var1
capture drop var2
capture drop var3
```

Example 3

Here is another example of using capture to dispose of nonzero return codes: When using do-files to define programs, it is common to begin the definition with capture program drop progname and then put program progname. This way, you can rerun the do-file to load or reload the program.

Example 4

Let’s consider programs whose behavior is contingent upon the outcome of some command. You write a program and want to ensure that the first argument (the macro ‘1’) is interpreted as a new variable. If it is not, you want to issue an error message:

```
capture confirm new variable '1'
if _rc!=0 {
    display "'1' already exists"
    exit _rc
}
(program continues. . .)
```
You use the `confirm` command to determine if the variable already exists and then condition your error message on whether `confirm` thinks ‘1’ can be a new variable. We did not have to go to the trouble here. `confirm` would have automatically issued the appropriate error message, and its nonzero return code would have stopped the program anyway.

> **Example 5**

As before, you write a program and want to ensure that the first argument is interpreted as a new variable. This time, however, if it is not, you want to use the name `_answer` in place of the name specified by the user:

```bash
capture confirm new variable '1'
if _rc!=0 {
    local 1 _answer
    confirm new variable '1'
}
(program continues...)
```

> **Example 6**

There may be instances where you want to capture the return code but not the output. You do that by combining `capture` with `noisily`. For instance, we might change our program to read

```bash
capture noisily confirm new variable '1'
if _rc!=0 {
    local 1 _answer
    display "I’ll use _answer"
}
(program continues...)
```

`confirm` will generate some message such as “...already exists”, and then we will follow that message with “I’ll use _answer”.

> **Technical note**

`capture` can be combined with `{}` to produce `capture blocks`. Consider the following:

```bash
capture {
    confirm var ‘1’
    confirm integer number ‘2’
    confirm number ‘3’
}
if _rc!=0 {
    display "Syntax is variable integer number"
    exit 198
}
(program continues...)
```

If any of the commands in the capture block fail, the subsequent commands in the block are aborted, but the program continues with the if statement.

Capture blocks can be used to intercept the `Break` key, as in
capture {
    stata_commands
}
if _rc==1 {
    Break key cleanup code
    exit 1
}
(program continues...)

Remember that Break always generates a return code of 1. There is no reason, however, to restrict the execution of the cleanup code to Break only. Our program might fail for some other reason, such as insufficient room to add a new variable, and we would still want to engage in the cleanup operations. A better version would read

capture {
    stata_commands
}
if _rc!=0 {
    local oldrc = _rc
    Break key and error cleanup code
    exit 'oldrc'
}
(program continues...)

Technical note

If, in our program above, the stata_commands included an exit or an exit 0, the program would terminate and return 0. Neither the cleanup nor the program continues code would be executed. If stata_commands included an exit 198, or any other exit that sets a nonzero return code, however, the program would not exit. capture would catch the nonzero return code, and execution would continue with the cleanup code.

Also see

[P] break — Suppress Break key
[P] confirm — Argument verification
[P] quietly — Quietly and noisily perform Stata command
[U] 18.2 Relationship between a program and a do-file