The Stata Journal (2014) 14, Number 1, pp. 1–2

# Stata tip #: Certifying subroutines

Maarten L. Buis
Wissenschaftszentrum Berlin für Sozialforschung (WZB)
Berlin, Germany
maarten.buis@wzb.eu

When writing your own program in Stata it is good practice and useful to create (and run) a certification script. A certification script is nothing other than a .do file that runs your program and compares the results with either some known to be true result or results from a previous run. Gould (2001) It is also useful and good practice to split-up your program into smaller subroutines, and you can store these subroutines in the same .ado file. These subroutines will only be visible to other programs defined within the same .ado file; the only program that is visible to all other programs in Stata will be the first program defined in an .ado file. This can be useful for subroutines that only make sense within the context of the main program. For example, one may want to delegate the parsing of some complicated syntax element to a subroutine. Moreover, putting a subroutine inside the .ado file of the main program protects users against accidentally running that subroutine. This can be important when, for example, the subroutine changes the data and the main program has various safeguards in place to ensure that this will not corrupt the user's data.

Sometimes it is helpful to certify some of the subroutines in isolation. How would one do that if the subroutines are not visible outside the .ado file? One could copy the subroutine and store it in its own file, thus making the subroutine globally visible. As mentioned above, there can be good reasons why one would not want the subroutine to be globally visible in the final program.

Another solution is to [R] **do** or [R] **run** the .ado file. This treats the .ado file as a regular .do file, which in this case only defines a set of programs. So after `do`ing or `run`ning an .ado file all its subroutines will also be available. Now one can certify the subroutines from the file that will be released to the general public without having to copy and paste parts out and into that file. This trick can also be useful when debugging a subroutine.

Consider the example .ado file below:

```
*! version 1.0.0 26Feb2014 MLB
program define mainprog
version 13
args input
subprog `input´
di `"`s(output)´"´
end

program define subprog, sclass
version 13
args input
sreturn local output `"do something smart with "`input´""´
end
```

 

If you store this file where Stata can see it (see [P] **sysdir**) or if you changed the working directory to where this .ado file is stored (see [D] **cd**), the command `mainprog` works directly, but if you try to call `subprog` Stata will return an error.

```
. clear all
. mainprog "this"
do something smart with "this"
. subprog "this"
unrecognized command:  subprog
r(199);
```

We can look at the names of the programs stored in memory using [P] **program dir**, and we see that `subprog` exists but only as part of the `mainprog` command.

```
. program dir
  ado      232  mainprog.subprog
  ado      213  mainprog
  (output omitted)
```

If we `run` this .ado file first then we can directly access both `mainprog` and `subprog`.

```
. clear all
. run mainprog.ado
. mainprog "this"
do something smart with "this"
. subprog "this"
. di `"`s(output)´"´
do something smart with "this"
. program dir
           232  subprog
           213  mainprog
  (output omitted)
```

## Reference

Gould, W. 2001. Statistical software certification. *Stata Journal* 1(1): 29–50.