Statistics/Data Analysis

fastcc

Title

fastcc -- ("fast copycode") run -copycode- with standard options

Syntax

fastcc [*projectname*] [**, { settings(***set_arg*[**:***set_value*]**)** | *fastcc_options* *copycode_options* **}**]

main_options	Description
projectname	Name of project to be processed
settings(*set_arg*[**:***set_value*]**)**	Record **fastcc** settings to ini file

fastcc_options	Description
alldepon(*depfilename***)**	Process all projects that use file *depfilename*
verbose	Show detailed **copycode** output instead of **fastcc**'s summary output

copycode_options	Description
nocopy	See copycode
simplemode	See copycode
force	See copycode
starbang(*stb_mode***)**	See copycode
noprogdrop	See copycode

Description

You must be familiar with copycode before you can understand what **fastcc** is doing.

fastcc has two main uses. First, it runs copycode with standard options so you do not have to type them at the command line.

Secondly, **fastcc** can re-process all projects that depend on one particular file. This is done by option **alldepon**.

Options

projectname will process project *projectname*.

settings(*set_arg*[*:set_value*]) displays and sets values recorded in **fastcc**'s ini file. This option is exclusive: You may not use it in conjunction with any other option.

The values of the path to the input file and to the target directory of the target file are stored in the text file `c(sysdir_plus)'/f/fastcc.ini (the "ini file"). If you wish, you can open and modify this file directly. Alternatively, issuing

. fastcc, settings(inputfile: *inputfilepathandname*)
. fastcc, settings(targetdir: *targetdirpath*)

will record your settings in the ini file. Each *set_arg* (inputfile or targetdir) must be followed by a colon and by a valid path to a file (inputfile) or directory (targetdir). **fastcc** assumes an output file name of *projectname*.ado which it will append to *targetdirpath*. Relative file paths are not allowed in the ini file. Settings arguments may not exceed 244 characters.

In addition,

```
. fastcc, settings(list)
```

will list the current ini file entries.

alldepon(depfilename) will process all projects that depend on file *depfilename*. Note that *depfilename* must be supplied with extension since **fastcc** must be able to identify the file among .mata, .stp, .adv, etc. files.

The interaction of this option with option **simplemode** is of particular importance. If you leave out option **simplemode**, **fastcc** runs through all projects in the input file and determines for each project whether the condition for ado mode are given (i.e. the project has a "main adv"). If so, it compiles a list of all first-order and higher-order dependencies; if *depfilename* is among them, it includes this project among the projects to be processed. If the conditions for ado mode for a particular project are not met, **fastcc** checks only the direct dependencies for the occurence of *depfilename*. If they inlcude *depfilename*, the project is added to the list of projects to be processed.

If you do use option **simplemode**, **fastcc** checks for each project only the direct dependencies for the occurence of *depfilename*.

If you use option **alldepon**, you cannot specify *projectname*.

verbose will display detailed output from **copycode** instead of the summary output from **fastcc**.

nocopy, **simplemode**, **force**, **starbang**, and **noprogdrop** fully correspond to **copycode** options and are passed to each **copycode** call made by **fastcc**.

Remarks

You can use **fastcc** to work more efficiently with copycode. **fastcc** runs copycode with standard options. These standard options are:

copycode option	***fastcc** equivalent*
inputfile	as recorded in **fastcc**'s ini file
targetfile	path join of target directory as recorded in **fastcc**'s ini file and the output file name, which is assumed to be the project name plus ado file extension
replace	is always assumed

That way, a copycode statement like

```
. copycode myproject, input(pathtoinput/input.txt)
  target(pathtotarget/myproject.ado) replace
```

conveniently collapses into

```
. fastcc myproject
```

If you are in the process of developing and testing an ado file and you are constantly making small changes and re-processing the project, this saves a good bit of typing.

The standard options are based on a set of assumptions that are typically true in practice:

- You are using only one input file that lists all of your projects. Very rarely will you have several versions of an input file or several input files for different purposes.
- The project names correspond to the names of the ado files you want to create.
- You know what you are doing when executing **fastcc** and it is ok to overwrite existing ado files.
- Your self-written ados reside in one directory (e.g. c:\ado\personal).

The second main purpose of **fastcc** is related to the following problem: Suppose you have made changes to a file called *depfilename* that goes into several of your projects. You do not know, however, the list of projects that directly or indirectly depend on *depfilename*. The statement

```
. fastcc, alldepon(depfilename) nocopy
```

will provide you with screen output on the list of projects in question. This list will also be saved in **r()**.

If you leave out option **nocopy** from the above statement, all of these projects will be re-processed. That way, after making changes to one particular file you can quickly bring all of your ado projects to the latest stage of your code development. After a succesful run of the certification scripts for the projects that have been re-processed (or, ideally, after running certification scripts on all your ado projects) you are all set.

--- Warning ---
There is no **replace** option in **fastcc**. Unless option *nocopy* is supplied, **fastcc** will always create or **overwrite** a file called *projectname*.ado. Similarly, when using option **alldepon**, it will create or **overwrite** files *projectname1*.ado, *projectname2*.ado, ..., where these file names correspond to the projects that have been processed. Be sure that you know what you are doing when using **fastcc**.

Examples

Before you can use **fastcc** you first have to set the default input file path and the target directory for output files. This requires running two statements (here: with bogus arguments)

```
. fastcc, settings(inputfile : c:\mypath\input.txt)
. fastcc, settings(targetdir : c:\mytargetdir)
```

Since they are stored in a file, these settings are kept in between Stata sessions and reboots. If you want to change these settings you simply re-issue these commands using different arguments. You can always query the current settings by

```
. fastcc, settings(list)
```

Continuing the examples from copycode, the command

```
. fastcc proj_a
```

is identical to the **copycode** statement of the first copycode example:

```
. copycode proj_a, inputfile(c:\mypath\input.txt)
    targetfile(c:\mytargetdir\proj_a.ado) replace
```

and will produce an identical outputfile. Note the implied **replace** option by the usage of **fastcc**.

Applying option **alldepon**,

```
. fastcc, alldepon(proj_a.adv)
```

is identical to the two **copycode** statements of the first copycode example and the second copycode example:

```
. copycode proj_a, inputfile(c:\mypath\input.txt)
    targetfile(c:\mytargetdir\proj_a.ado) replace
. copycode proj_b, inputfile(c:\mypath\input.txt)
    targetfile(c:\mytargetdir\proj_b.ado) replace
```

and will produce identical output files. This is because both proj_a and proj_b depend on proj_a.adv.

Saved results

fastcc saves the following in **r()**:

Macros

When supplying *projectname*, results saved will correspond exactly to the ones returned by **copycode**.

When using option **alldepon**, saved results are instead:

r(projectlist)	List of projects that depend on *depfilename*. Contains tokens from r(failed_make) but not from r(failed_check).
r(failed_make)	List of projects that could not be processed because errors occured
r(failed_check)	List of projects that could not be checked for dependencies because errors occured

All returned values are lower case, irrespective of what the actual casing of the files on disk or in the input file is.

Author

Daniel C. Schneider, Goethe University Frankfurt, schneider_daniel@hotmail.com

References

Schneider, D.C. (2012). Modular Programming in Stata. Presentation at the German Stata Users Group Meeting 2012, Berlin. Available at