Modular Programming in Stata

German Stata Users Group Meeting

WZB Social Science Research Center, Berlin
June 2012

Daniel Schneider
Goethe University Frankfurt
daniel.schneider@wiwi.uni-frankfurt.de

A word of caution about terminology

- In this talk, I define and use various terms from computer science you may have heard before: modular programming, assembly, project, etc.
- However, I am an economist, not a computer scientist, and may not be aware of the fine implications of some terms.
- Terms used in this talk should only be understood within the context of this talk: the exposition of the user-written command -copycode- which provides a framework for writing Stata code.
- Suggestions for improvements in terminology are very welcome.

Programming and distributing Stata command additions

- In addition to writing scripts („do-files“), Stata allows for the programming of new Stata commands („ado-files“).
- These can be as powerful as official Stata commands.
- You can easily distribute your self-written commands as so-called „packages“ through a web site.
  For instructions, see -help net- and -help usersite-.
- Any Stata user can then „install“ your files using -net install-.

Programming and distributing Stata command additions

- The installation process is quick, easy, robust:
  - Requires execution of only one command:
    . net install *packagename*, from(*URL*)
    See also: -help ssc-
  - No manual download or manual moving of files.
  - No changes to the adopath are required.
  - Since typically very few files are distributed per package, function (ado-file) name clashes are unlikely.
- Distributing user-written code in Stata is much easier than it is in many alternative software packages, especially matrix language based applications.

Motivation for writing -copycode-

- When I started using Stata, I wrote many ado-files that specifically served my needs.
- When I wrote a new command, I freely used calls to commands I had previously written, or other user-written commands. I built up a ***system of ado-files***.
- I ended up having 50+ ado-files that were mutually dependent on each other.
- When I wanted to distribute my commands, I found that dependencies were so complex that distribution was extremely cumbersome.

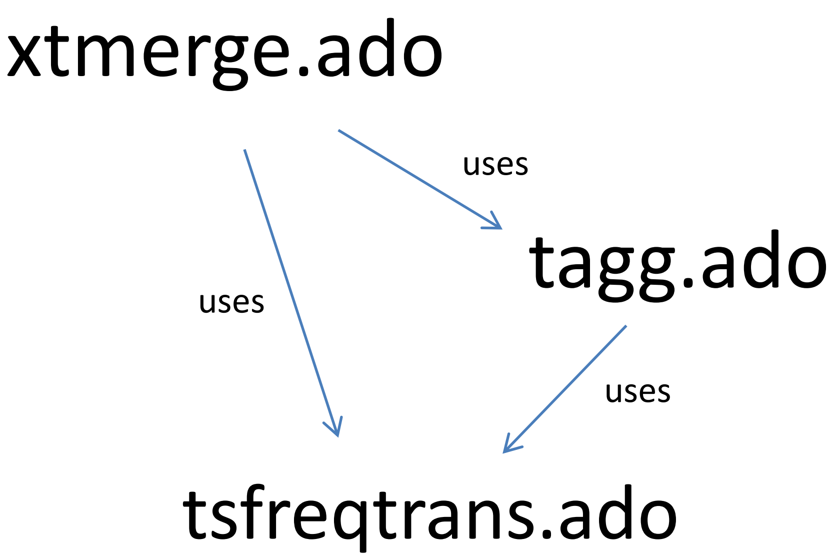
System of ado-files

Example:
time series / panel data:
temporal aggregation and merging panel datasets

- -tagg- command: temporal aggregation of time series
  - uses -tsfreqtrans-:
    convert frequency encodings (e.g. m-Monthly-12)
  - see also Christopher Baum's -tscollap- on SSC
- -xtmerge- : merging panel datasets
  - uses -tagg- command:
    harmonize frequencies of two datasets before merging
  - also uses -tsfreqtrans-

System of ado-files

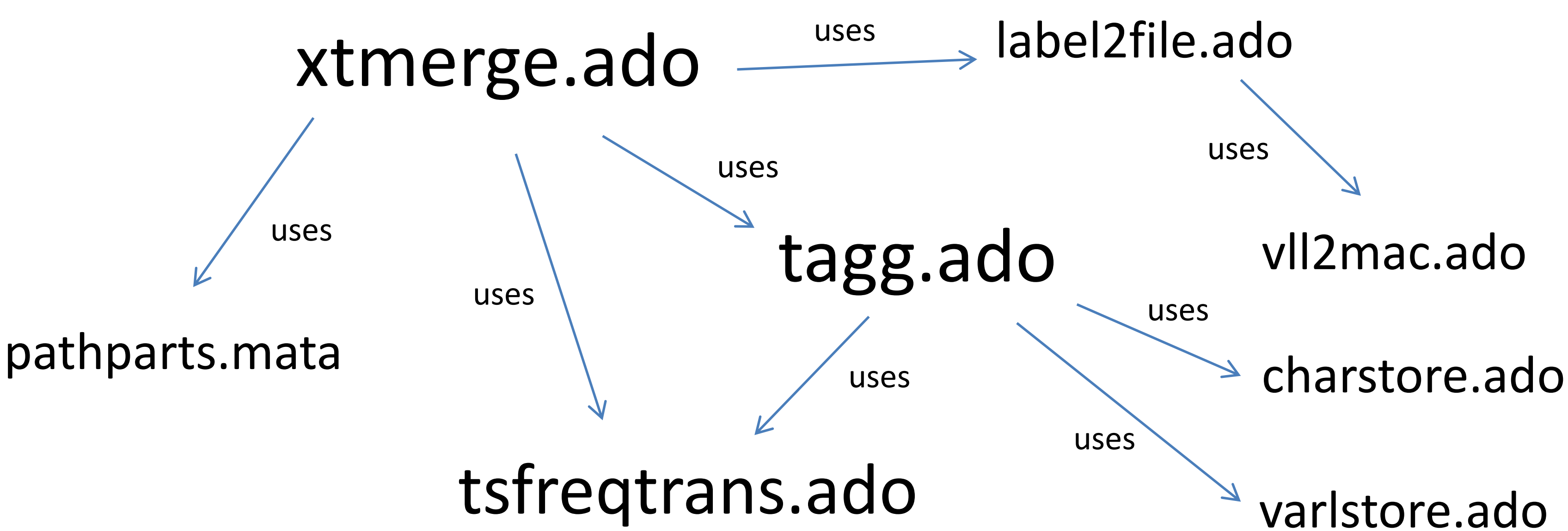
Simple example:
time series / panel data:
temporal aggregation and merging panel datasets



Terminology
A *dependency* is a function or ado-file that is called by another function or ado-file. The latter is said to have dependencies. There can be direct / first-order and higher dependencies.

System of ado-files

Expanded example:
time series / panel data:
temporal aggregation and merging panel datasets



Code production vs. code distribution

- There is a conflict between the requirements of efficient programming and easy distribution.
- As a programmer, you want to re-use previously written code in different contexts by creating your ***system of ado-files and subroutines***.
- The benefits are:
  - Code production
    Code re-use reduces the number of lines that need to be written.
  - Code certification
    When writing a new ado-file, functionality provided by calls to existing ado-files and subroutines does not have to be tested again.
  - Code maintenance
    Fixing bugs / enhancing subroutines only has to be done in one place.

Code production vs. code distribution

- Having a system of ado-files means that most self-written ado-files depend on many other self/user-written ado-files.
- This causes problems for ado-file distribution:
  - It is difficult to find out the list of self/user-written ado-files that a self-written ado-file potentially uses.
  - ado-file naming conflicts become more likely.
- Therefore, for distribution *modular* files are desirable, i.e. code that is self-contained, i.e. non-dependent on code outside of the file.

<= Terminology: modular ado-file

Problem: How accommodate dependencies when distributing code?

Solution candidate I: Distribute large packages?

- -xtmerge- package contains 8 code files.
- Difficult to find out the list of self-written subroutines / ado-files that an ado-file uses.
- ado-file name clashes
- May require to include ado-files in the package whose functionality has nothing to do with the one from the main ado of the package.

Problem: How accommodate dependencies when distributing code?

Solution candidate II: Separate distribution?

- For our simple example, we would create separate packages for -tagg- and -xtmerge-.
- The package description file for -xtmerge- would indicate that the installation of -tagg- is a prerequisite.
- This strategy is worse. Uninstalling -tagg- would break -xtmerge-.

Problem: How accommodate dependencies when distributing code?

Solution candidate III: copy & paste?

- Copy & paste code from tagg.ado, tsfreqtrans.ado into distribution version of xtmerge.ado.
- Progress: We now can distribute modular code.
- We still have the problem of having to figure out dependencies.
- Copy & paste must be done carefully to avoid multiple function definitions.
- Copy & paste is time-consuming, error-prone, inefficient.

Problem: How accommodate dependencies when distributing code?

Proposed solution:

- Write new Stata command whose main features are that it
  - provides an algorithm for detecting unique dependencies of all orders.
  - automates copy & paste operations.
- I named this command -copycode-.

-copycode- design stage: things to consider

- How does -copycode- know about the subroutines that a new ado-file uses?
- What do we do with second-order and higher dependencies?
- How do we prevent accidentally overwriting newly written code?
- Do we always want to copy the entire source files, or just sections thereof? In the latter case, how do we implement this?
- How do we prevent duplicate copying of subroutines? Do we have to worry about subroutine name clashes?

-copycode- design: outline of usage

1) For each ado project, make a list of ado/mata functions that this project *directly* depends on.
2) Make this list for all ado projects you have worked on and put them into one big input list for -copycode-.
3) For any particular project, -copycode- uses the information contained in the input list to figure out a complete list of first-order and higher dependencies for that project.
4) It will copy the code from these files into a target (ado) file.

⇒ copycode necessitates the separation of files that contain the code typed by you, the programmer, and files whose code is executed.

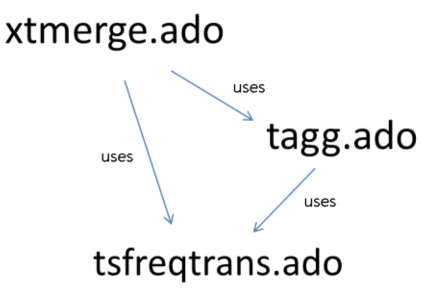
⇒ use file extension .adv ("ado development") for writing code.

To avoid accidentially overwriting typed code,
never ever edit ado-files; instead, edit adv-files.

- copycode- input list: simple example

Contents of ccinput_simple.txt:

```
// simple example input list for -copycode-
// German Stata Users Group Meeting 2012

xtmerge   c:\mypath\xtmerge.adv
xtmerge   c:\mypath\tagg.adv
xtmerge   c:\mypath\tsfreqtrans.stp

tagg     c:\mypath\tagg.adv
tagg     c:\mypath\tsfreqtrans.stp
```



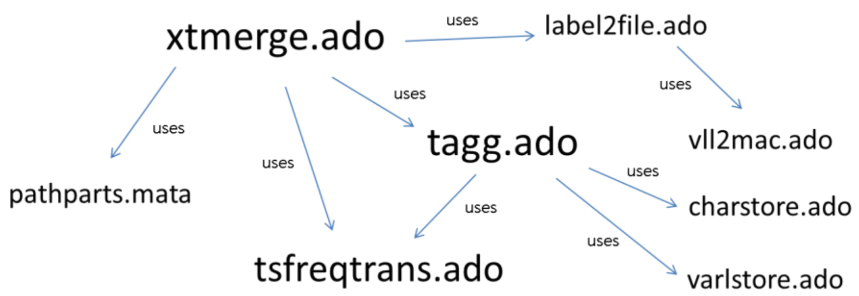
Terminology

- Project:
  entries in first column
  of -copycode- input list
- Source files:
  entries in second column
  of -copycode- input list
- Main adv:
  first entry of a project (if it is
  an adv-file)

- copycode- input list: expanded example

Contents of ccinput_expanded.txt:

```
// expanded example input list for -copycode-
// German Stata Users Group Meeting 2012

xtmerge   c:\mypath\xtmerge.adv
xtmerge   c:\mypath\tagg.adv
xtmerge   c:\mypath\tsfreqtrans.stp
xtmerge   c:\mypath\label2file.adv
xtmerge   c:\mypath\ds_pathparts.mata

tagg      c:\mypath\tagg.adv
tagg      c:\mypath\charstore.adv
tagg      c:\mypath\varlstore.adv
tagg      c:\mypath\tsfreqtrans.stp

label2file   c:\mypath\label2file.adv
label2file   c:\mypath\vll2mac.adv

vll2mac     c:\mypath\vll2mac.adv

charstore    c:\mypath\charstore.adv

varlstore    c:\mypath\varlstore.adv
```



I use the ***extension .stp*** („Stata program files“) for text files that contain simple Stata programs and are used as subroutines by different ado-files („common subroutines“).

-copycode- syntax

copycode , <u>inp</u>utfile(*inputfilename*)
<u>pr</u>oject(*projectname*)
<u>t</u>argetfile(*targetfilename*)
[replace <u>si</u>mplemode force]

Executing

```
. copycode , inputfile(c:\mypath\ccinput_simple.txt)
  project(xtmerge) targetfile(c:\mypath\xtmerge.ado) replace
```

produces the screen output

copied: c:\mypath\xtmerge.adv
copied: c:\mypath\tagg.adv
copied: c:\mypath\tsfreqtrans.stp
project xtmerge successfully assembled
outputfile: c:\mypath/xtmerge.ado

Terminology
To **assemble** a project means to use -copycode- to produce an output file for a certain project.

Dependencies and file types

-copycode- handles source file entries with the following file extensions in a special way

- .adv ("ado development file"):
  must have a corresponding project; its dependencies are added to the list of files to be included in the target file.
- .ado
  treated as a modular user-written ado-file
- .stp ("Stata program file“)
  treated as a file that defines a (small) modular user-written Stata program
- .mata
  currently treated as a modular Mata function definition file.

The order of occurrence of source file contents in the target file is:

adv-files => ado-files => stp files => mata-files => other files

Copy regions

- An additional feature of -copycode- is that within each source file it allows to switch copying on and off. -copycode- looks for the (comment) lines that contain

  !copycodebeg=>

  !copycodeend||

  and switches copying on and off accordingly.
- Lines containing these strings are called copy region limits. The code in between these lines is called a copy region. A file can have multiple copy regions.
- Benefits:
  - private vs. public comments
  - crucial for designing effective development and certification file templates. See Gould (2010), pp. 14-18.

-xtmerge- example code outline: main adv

Contents of xtmerge.adv:

```
// !copycodebeg=>
*! version 0.0.1  1jun2012  dcs

program define xtmerge
                (…code goes here…)
               quietly tagg …   // <= call to tagg
               quietly tsfreqtrans … // <= call to tsfreqtrans
/*    !copycodeend||
   any text that goes here serves as a “private” comment and will not be added to the output file
     !copycodebeg=>    */
               (…more code goes here…)
end

program define xtmerge_sub1
               (…)
end
// !copycodeend||

any text that is below the last „!copycodeend||” occurrence will remain “private”; e.g. ToDo list, version history, possible enhancements
```

-xtmerge- example code outline: adv dependencies

Contents of tagg.adv:

```
// !copycodebeg=>
*! version 0.0.1  1jun2012  dcs

program define tagg
                (…code goes here…)
                quietly tsfreqtrans … // <= call to tsfreqtrans
/*    !copycodeend||
   again, switch copying on and off if desired
     !copycodebeg=>    */
                (…more code goes here…)
end

program define tagg_sub1
                (…)
end
// !copycodeend||
```

-xtmerge- example code outline: stp dependencies

Contents of tsfreqtrans.stp:

```
// !copycodebeg=>
// version 0.0.1  1jun2012  dcs

program define tsfreqtrans
        (…code goes here…)
end
// !copycodeend||
```

-xtmerge- target file creation

. copycode , inputfile(c:\mypath\ccinput.txt) project(xtmerge) targetfile(c:\mypath\xtmerge.ado) replace

Contents of xtmerge.ado (continued on next page):

```
*! version 0.0.1  1jun2012  dcs

program define xtmerge
        (…code goes here…)
        quietly tagg …   // <= call to tagg
        quietly tsfreqtrans … // <= call to tsfreqtrans
        (…more code goes here…)
end

program define xtmerge_sub1
        (…)
end
```

-xtmerge- target file creation

Contents of xtmerge.ado (continued from previous page):

```
// *! version 0.0.1  1jun2012  dcs

program define tagg
		(…code goes here…)
		quietly tsfreqtrans … // <= call to tsfreqtrans
		(…more code goes here…)
end

program define tagg_sub1
		(…)
end

// *! version 0.0.1  1jun2012  dcs

program define tsfreqtrans
		(…code goes here…)
end
```

Downsides of using -copycode-

- code bloat
- somewhat complicated
- limited ability to format target ado-file

Fast -copycode-: -fastcc-

- -fastcc- is a wrapper function for -copycode- that runs it with standard options (e.g. the standard input list).
- It was written because writing and debugging commands requires very frequent re-assembly of projects.
- Has option -alldepon-: re-assembles all projects that directly/indirectly depend on a certain file.

Miscellaneous remarks

- starbang lines
- See also the -adolist- package: Jann (2007)
- version statements
- subroutine name clashes
- reverting back to a non-modular system
- other uses: option -simplemode-
- -copycode- can be used to get a list of files to be included in „traditional“ multiple-file packages.
- One can easily include ado-files written by other users.

Thank you

References

Gould, William (2010): Mata, the Missing Manual. Presentation at the UK Stata Users Group Meeting 2010, London. Available at http://www.stata.com/meeting/uk10/UKSUG10.Gould.pdf

Jann, Ben (2007): Adolists – A New Concept for Stata. Presentation at the UK Stata Users Group Meeting 2007, London. Available at http://repec.org/usug2007/London07_adolist.pdf

Acknowledgments

I thank Kevin Crow from StataCorp for helpful comments.