The Stata Journal

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata and Mata are registered trademarks of StataCorp LP.

The Stata Journal (2007)
7, Number 1, pp. 98–105

File filtering in Stata: Handling complex data formats and navigating log files efficiently

John Eng
Department of Radiology and Radiological Science
Johns Hopkins University School of Medicine
Baltimore, MD
jeng@jhmi.edu

Abstract. A text file filter is a program that converts one text file into another on the basis of a set of rules. For statistical applications, a text file filter can convert data embedded in a complicated text file so that Stata can read and analyze it. A text file filter can also automate the production of more user-friendly output from long Stata log files. The `file` command lets you use text file filters in Stata. This article reviews some key programming points for successful implementation of such filters.

Keywords: dm0027, nullfilter, matchfilter, hyperlog, file, log, navigation, text, filtering, data import, data export, HTML, hyperlink index

1 Introduction

File filters are a class of programs that finds use in many computing environments. The typical file filter reads a text file line by line, performs an operation on each line, and writes the resulting text to an output text file.

This article considers three examples of file filters that may interest Stata users. The first filter, `nullfilter`, does nothing but copy the input file, but it provides a complete template on which you can build useful filters. It also provides an opportunity to review a few disparate and possibly underappreciated programming points that are nevertheless key to successful implementation of filters in Stata.

The second filter, `matchfilter`, demonstrates how even a simple filter can allow Stata to import data that are embedded in a file whose format is too complex to be expressed in a dictionary file. Finally, the third filter, `hyperlog`, starts with a Stata log file and generates a web browser–compatible hyperlinked index that improves the efficiency of navigating the log file. Improving the navigation of potentially long log files will probably interest many Stata users, so feel free to skip directly to the description of `hyperlog` if the underlying principles of file filtering do not interest you.

2 Example 1: nullfilter

File filters are probably easiest to implement in scripting languages (e.g., Perl) that have been optimized for text processing. But using such a program would be cumbersome

because it would not be integrated into Stata. There are also computer platform dependencies (e.g., differences in new-line characters) to worry about in other languages. If you remember a few key programming points, implementing a filtering program in Stata actually is only slightly harder than in some other languages, although a Stata implementation is much easier to use for those working in Stata. Also, cross-platform differences in new-line characters are handled within Stata and are not a programming concern.

The most basic file filter is one that performs no operation and simply copies the input file:

```
program nullfilter
        version 8.0
        args inputFileName outputFileName dummy

        ** Check for correct number of arguments
        if (("`inputFileName'" == "") | ("`outputFileName'" == "") /*
                */ | ("`dummy'" != "")) error 198

        ** Open input and output files
        tempname inputFileHandle
        tempname outputFileHandle
        file open `inputFileHandle' using "`inputFileName'", read text
        file open `outputFileHandle' using "`outputFileName'", write text

        ** Null file filter
        local lineCount = 0
        file read `inputFileHandle' textLine
        while (r(eof) == 0) {
                * File filter code goes here
                file write `outputFileHandle' `"`macval(textLine)'"' _n
                local lineCount = `lineCount' + 1
                file read `inputFileHandle' textLine
        }

        ** Clean up
        file close `inputFileHandle'
        file close `outputFileHandle'
        display as text "Copied `lineCount' lines."
end
```

In Stata, file filters rely on the `file` command, which was introduced in Stata 8. The command is well described in the Stata documentation (see [P] **file**), and authors have presented examples to illustrate its usefulness in generating files for other applications (Slaymaker 2005).

The null file filter's program structure forms the basis for `matchfilter` and `hyperlog`. All three programs demonstrate several key points to remember when implementing file filters in Stata.

2.1 Temporary names for file handles

Files opened with `file open` may remain open if the program aborts abnormally, which may happen during program development or unanticipated situations after deployment. If the file handle is created using `tempname`, Stata will close the file automatically when the ado-file ends (normally or abnormally). This feature would always be desired in a normal application:

```
tempname inputFileHandle
file open ‘inputFileHandle’ using "‘inputFileName’", read text
tempname outputFileHandle
file open ‘outputFileHandle’ using "‘outputFileName’", write text
```

2.2 Proper program looping

Stata does not appear to set the `r(eof)` end-of-file flag until after an attempt is made to read past the end of the file. Therefore, two file read commands are needed, one outside the beginning of the `while` loop and one inside:

```
file read ...
while (r(eof) == 0) {
        .
        .
        .
        file read ...
}
```

2.3 Compound double quotes

When creating or manipulating macros that contain arbitrarily complex text, such as HTML, realize that such text can and often does contain double quotes, single quotes, dollar signs, or other characters that Stata may consider *special*. Therefore, enclosing such macros in compound double quotes is prudent; see [U] **18.3.5 Double quotes**. Using the `macval()` macro expansion function is important to prevent interpretation of any embedded text that might look like a macro (see [U] **18.3.7 Macro increment and decrement functions**):

```
file write ‘outputFileHandle’ ‘"‘macval(textLine)’"’ _n
```

Here the macro `textLine` contains the text to be written to the file. Not taking these precautions can cause nonspecific and potentially misleading error messages. Although compound double quotes are probably a rare finding in most Stata programs, they should be used copiously in file filters written for Stata. When accessing the value of a macro that contains arbitrary text, the programmer should assume that both compound double quotes and the `macval()` function are needed unless there is a specific reason to do otherwise. This practice is important not only for instances of the `file` command but also for any statement that manipulates the macro's value.

2.4 String length limitation

If a file filter uses any Stata string functions, they are limited to 244 characters (80 in earlier versions). A file filter using these functions would truncate filtered lines longer than the maximum length. Fortunately, macro-extended functions (see [U] **18.3.4 Macros and expressions**) are available and do not have this limitation. For example, a file filter producing an HTML output file might want to be fully compliant with HTML standards by properly encoding all ampersands occurring in a macro:

```
local s = subinstr(‘"‘macval(s)’"’, "&", "&amp;", .)
```

However, this filter would silently truncate the text at 244 characters. A better alternative would be to use the equivalent macro-extended function:

```
local s: subinstr local s "&" "&amp;", all
```

The character limit also has a subtle effect on macro definition:

```
local s2 = ‘"‘s1’"’
local s2 ‘"‘s1’"’
```

The first statement evaluates `‘"‘s1’"’` as a string expression before assigning the value to `s2`. Therefore, the string length limit applies, and the macro assignment may result in truncation. The second statement specifies a macro copy operation that is not subject to the string length limit.

2.5 Tab characters

Stata considers tabs and spaces to be different characters (as they are). This distinction has consequences if the file filter uses certain string functions (`trim`, `ltrim`, `rtrim`, `word`, `wordcount`) to process an input file containing tabs. The `trim` functions do not strip leading or trailing tabs, and words separated by tabs are not considered separate words. In this situation, these functions will work as expected if you first replace all tabs with spaces:

```
local s = subinstr(‘"‘macval(s)’"’, char(9), " ", .)
```

This substitution is subject to the string length limitation.

2.6 Other general considerations for Stata file filters

In Stata, some file filters require no programming at all. Stata's built-in `filefilter` command (see [D] **filefilter**) searches and replaces simple string patterns in a file. This command is adequate for many common tasks (e.g., replacing all tabs in a file with spaces).

Stata 9 introduced Mata, a new compiled programming language within Stata that has a syntax similar to the C programming language. Mata has file I/O and string

functions that resemble those in the standard C library, so Mata would be well suited for writing file filters. Because Mata is compiled, you can expect programs written in it to run fast. However, the fundamental program logic of the file filter would be the same.

3 Example 2: Simple match filter

Stata's `infile` (see [D] **infile (free format)** and [D] **infile (fixed format)**) and `insheet` (see [D] **insheet**) commands can handle many data input formats, but they cannot extract data from sources with complex patterns. Suppose that we wish to analyze the time required to resolve computer support problems according to their priority level. We have a log file from a problem tracking system:

```
************************************************************************
Number:    8241
Title:     Intermittent delays in new workstation version
Priority:  Normal

10/6/2006 11:39 AM - Chris - Resolved
DNS settings were not populated on this machine causing several issues.  After
we entered in all the new DNS settings, the machine worked normally.

10/6/2006 9:29 AM - Tom - Assign
Please describe which workstation was involved and any other software you were
using at the same time.

10/5/2006 7:42 PM - User - Create
Throughout the day in my office, the workstation had intermittent 5-10 second
delays.

************************************************************************
Number:    8228
.
.
.
```

We are interested in the priority, creation date–time, and resolution date–time for each problem. Because each observation can span a different number of lines, we must first filter the file before it can be imported into Stata with `infile`. The filter copies only the lines of interest and can be derived from `nullfilter` by enclosing the `file write` statement inside an appropriate conditional:

```
if ((index(`"`macval(textLine)'"', " - User - Create") > 0) /*
        */ | (index(`"`macval(textLine)'"', " - Resolved") > 0) /*
        */ | (index(`"`macval(textLine)'"', "Priority:  ") == 1)) {
        file write `outputFileHandle' `"`macval(textLine)'"' _n
        local lineCount = `lineCount' + 1
}
```

For the first observation, this method produces the following output:

```
Priority:  Normal
10/6/2006 11:39 AM - Chris - Resolved
10/5/2006 7:42 PM - User - Create
```

This output is readable by `infile` with an appropriate dictionary:

```
dictionary {
        _lines(3)
        _line(1)
                _column(12)
                str20 priority %s
        _line(3)
                str20 cdate %s
                str20 ctime %s
                str20 campm %s
_line(2)
                str20 rdate %s
                str20 rtime %s
                str20 rampm %s
}
```

After `infile`, a series of Stata commands would be required to convert the strings into numerical date and time values.

Stata 9 introduced new functions that support *regular expressions*. Regular expressions are a notation system for expressing complex text patterns that includes a system of wildcard matching (Turner 2005). These functions allow the matching rule of a file filter to be based on something more sophisticated than a simple identical match.

4 Example 3: Generating a hyperlinked index for browsing log files

A Stata do-file can generate a lot of output, so capturing this output in a log file is convenient. But wading back and forth through a long log file to review and collect results can be time consuming, especially if the do-file contains many steps or sequences that look similar. Even when a do-file is organized and documented well, the sheer size of the corresponding log file can be disorienting.

A well-known principle for creating effective user interfaces is the visual information–seeking mantra, "overview first, zoom and filter, then details-on-demand" (Shneiderman 1996). If we were to apply this mantra to improve log file navigation, we might envision a document in which the do-file is displayed as the overview, and clicking a line in the do-file would display details from the corresponding log file (see figure 1).

(*Continued on next page*)

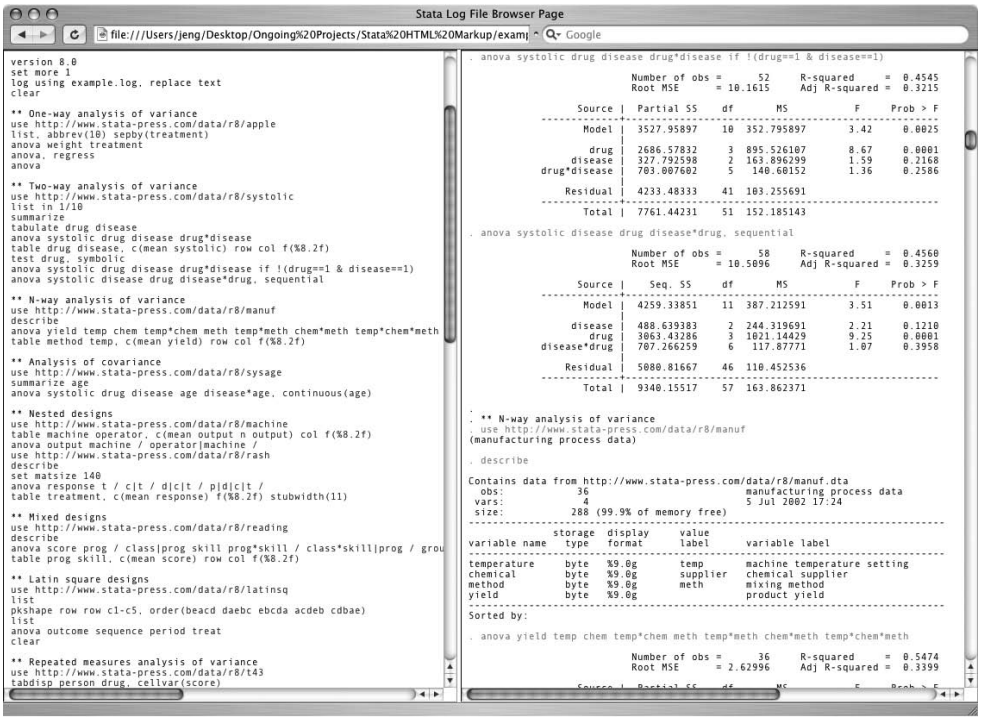

Figure 1: Hyperlinked index of a Stata log file

When a do-file is run with logging enabled, every line in the do-file appears somewhere in the log file, and Stata's results appear underneath. In theory, a program should be able to scan the two files for matching lines and, when found, add HTML hyperlink tags (`<A>`) to link the matching lines. The program `hyperlog` implements this idea as a file filter.

Actually, `hyperlog` consists of two file filters running in tandem (see figure 2). The first filter builds an overview file by reading the do-file one line at a time, looking for Stata commands (lines starting with a period). A second filter builds a details file by processing the log file that the do-file generated. For each command line found by the first filter, the second filter is activated and scans the log file for a matching command line. When a match is found, HTML hyperlink tags are added to both the overview and details files. For noncommand lines, both filters simply copy the text into their respective output files. In addition to these two output files, hyperlog generates a master HTML document that contains appropriate HTML `<FRAME>` tags to display the overview and details files side by side in any web browser.

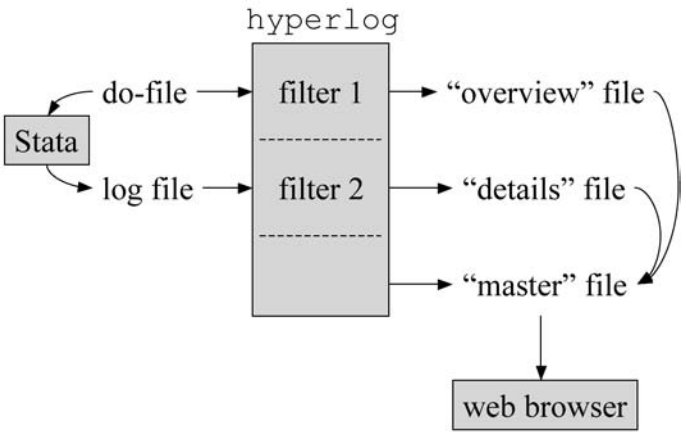

Figure 2: Processing of files to create a hyperlinked index

5 Conclusions

The `file` command lets Stata output results in other text formats, such as HTML (Slaymaker 2005) and even XML. This ability obviously adds flexibility when Stata results are intended for processing by other programs. The `file read` command also lets Stata directly read data files in arbitrary formats. This article discussed file filters that can perform complex operations on text files generated by Stata and other sources. The reading of external binary data files is probably too complex to be worthwhile for most Stata programmers, but the conversion of text files may often be worthwhile. When processing arbitrary text files, you must take care to properly handle *special* characters and potentially long lines of text.

6 References

Shneiderman, B. 1996. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Conference on Visual Languages*, 336–343. New York: IEEE.

Slaymaker, E. 2005. Using the file command to produce formatted output for other applications. *Stata Journal* 5: 239–247.

Turner, K. S. 2005. FAQ: What are regular expressions and how can I use them in Stata? http://www.stata.com/support/faqs/data/regex.html.

About the author

John Eng is a statistically inclined radiologist and associate professor in the Department of Radiology and Radiological Science at the Johns Hopkins University School of Medicine. His research interests are in clinical epidemiology, evidence-based radiology, and health sciences informatics.