

The River™  
Tiny Recombinant Processing Modules  
Release 2.1

Lower Cape Cod Digital

1988-2007. River Core, Rcore (tm) trademarks of Lower Cape Cod Digital

19 Old Chequessett Neck Drive, Wellfleet, MA 02667  
www.lccdigital.com support: support@lccdigital.com

The following provision does not apply to any country or locality where such provisions are inconsistent with local law. LCCDigital Software Group provides this publication on an "as-is" basis without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Some localities do not allow disclaimer of express or implied warranties for certain transactions, therefore this statement may not apply to you.

## **Introduction**

The River is a high level streaming environment using pipes and batch facilities. It enables pluggable pieces to perform many tasks that are awkward with integrated GUI environments.

### System Requirements To Run River

Win32 operating systems.

### Licensing

GPL Licensing.

### Background

The basic assumption of streams is characters. The basic assumption of the River is lines of characters, strings.

Many tasks have to do with creating, manipulating and managing lines intelligently. File management, backups, web page creation, database management and email, to name but a few that the River has accomplished with greater speed and efficiency than the usual approaches.

The River is made up of hundreds of tiny, recombinant programs

Rivers and stream software architectures add

1. Ease of development:
2. Reliability and security, .easy to test intermediate results..
3. Great flexibility, recombinant procedures inherently lend themselves to a wide variety of tasks.
4. Efficiency of learning as .

## Contents

### **Introduction 2**

*System requirements to run River 2*

*Licensing 2*

*Background 2*

### **Installing and Running River 6**

### **River and Streams 7**

*Streams 7*

*Rivers 7*

### **Using the River 8**

*Getting Command "Usage": 8*

### **River Procedure Files 11**

*2.bat 11*

*Tabl2Frm.bat 12*

*DelTemps.bat 12*

*DelCache.bat – delete cache files 13*

*Ordinary pipes 14*

*Non-Cache Cache Directories 14*

*BigFiles.bat – List the biggest files on the system 15*

### **River Application Kits 18**

*Apache Access Logs Application Kit 18*

*Processing Logs ProcLog 1 to 5 19*

*Switching Log Files 20*

*Proclog1 Annotated 20*

*ProcLog1 – Only IP 22*

*Or, straight from the log file 22*

*Other ProcLog Batch Files 22*

### **Miscellany 23**

*XP command screens 23*

*Internal Line Lengths 23*

*Parameter Counts and processing 23*

*Indirect Parameters 23*

*Function Lists 23*

*River Size Capabilities 23*

### **The Lexecon 24**

## **Release 2.1 River Modules List 25**

*Addcols* – add columns to the River 26  
*CDD* – Change Directory Down 26  
*Choosfil* – User file selection 26  
*Choositg* – Select an item from the River 26  
*Clip2Fil & Clip2Riv* 27  
*Count and Counter* 27  
*Columnit* – Place the River in columns for display or reports 27  
*Delwith and Delwithi* 27  
*Doit* 28  
*Dropdriv, Dropext, Dropfile and Droppath* 28  
*dupelim* eliminates duplicate lines 28  
*Dupenumb* numbers duplicate lines 28  
*Endit* – terminate a batch file 29  
*Exist* filename command 29  
*Fgetattr, fgetcrea, fgetlast, fgetmod, fgetsize* 29  
*fgetmd5, fgetshal* 29  
*Fil2Clip* – Send the Clipboard to a file 29  
*Filetail* keeps the back end of a file's data 29  
*Fsetattr, fsetcrea, fsetlast, fsetmod, and fsetsize* 30  
*H2adate H2adatef* – Hex to alpha dates 30  
*ifk* – Insert File and kill it 30  
*ifl* – Insert File 31  
*Insleadb* – Insert Leading Blanks 31  
*Keep* – Keep lines in the river 31  
*kepbot & keptop* -- keep the bottom or top of the river 31  
*Kepwith & kepwithi* 31  
*Lex* – edit a lexecon item 31  
*Lexget & lexput* 31  
*lertext* – Show the items in the Lexecon file 32  
*listdirs* – List directories like... 32  
*listfile* – List files like 32  
*maxscr* – Maximize the screen in a command window 32  
*Prepend* -- Introduce a file to the beginning of the River 32  
*Prepline* – prepare River Lines 32  
*Postpend* -- Add a file to the end of the River 33  
*retcd* – show the return code for a program 33

*Riv2clip* – send the river to the clipboard 33  
*saveas* – save the River to a user selected file name 34  
*selectab* – Select Tabbed Columns 34  
*setcwd* – Set the current working directory 34  
*shortnam* -- Convert file or path names to 8.3 34  
*snip* – take segments out of lines 35  
*Snipfrom* – take a segment starting with a string 35  
*Snipto* – take a segment up to a string 35  
*Tabonspc* 35  
*ToglScr* – Toggle full screen mode for cmd windows 35  
*Usage* – quick Documentation 36

### **The Lexecon™ 37**

*Lexecon Format* 37  
*The Count and Counter Lexecon Section* 37

### **Windows Problems and Warnings 38**

#### **Warning on Parameters for Batch Files 39**

*Windows errs on some versions* 39  
*Work Around* 39

#### **NUL: Windows discrepancy 40**

*NUL: vs NULL:* 40  
*Work Around* 40

## **Installing and Running River**

1. Pickup the River zip file from SourceForge. <http://www.sourceforge.net/projects/river>
2. Make a new directory.
3. Unzip the files to the new directory.
4. You may want this directory to be part of the Path, please see web searches or your OS documentation on the PATH if you are not familiar with the path environment variable.

If you have problems, suggestions or bug reports, please e.mail a description of the difficulty to [support@lccdigital.com](mailto:support@lccdigital.com)

## River and Streams

### Streams

*Streams* and *Pipes and Filters* are powerful architectural tools for systems design and technical use. They have many features; one can easily learn a lot more with web searches.

### Rivers

*Rivers* are different from streams in that they expect ASCII lines instead of binary characters.

Purists might say this is a limitation, however it is also a simplification and Rivers are darn useful because so much of what happens in a system having to do with the user, command and control, and communications can be represented effectively in ASCII lines.

Consider that the following items use ASCII lines and consider the breadth of tasks involved:

1. Error Logs for Internet servers.
2. Name and Address date.
3. Commands for command line interpreters.
4. File and path information.
5. Configuration files for servers and FireWalls.

## Using the River

### Getting Command "Usage":

If you enter "/" as a parameter for any command it will show a usage definition screen.

For any River program you may also enter

```
usage program
```

where *program* is the name of the program. Usage is a program that consults a file named *Lexecon* that has definitions of the programs' use.

To see the list of programs enter

```
lextitle
```

You will see a list of the titles, as of 3.4.07:

```
Titles in E:\rivera\lexecon.
```

```
Count  
DelCache  
Dels  
deltop  
DelWith  
detritus  
doit  
DropDriv  
DropExt  
DropFile  
DropPath  
dupelim  
DupeNumb  
FAddAttr  
FaddCRC  
faddcrea  
faddmd5  
FAddLast  
faddsha1  
FAddMod  
Faddsize  
FileTail  
funique  
H2ADate  
insleadb  
Keep  
Keptop  
Kepwith  
Lex  
Lexput  
listfile  
listdirs  
Params  
Pass
```

Or, if you wish them in columns, enter

```
lextitle | columnit
```

to see:

```
E:\rivera>lextitle | columnit
Titles in E:\rivera\lexecon.

Count    DelCache Dels      deltop    DelWith   detritus doit
DropDriv DropExt  DropFile DropPath dupelim  DupeNumb exist
FAddAttr FaddCRC  faddcrea faddmd5  FAddLast faddshal FAddMod
Faddsize FileTail funique  H2ADate  insleadb Keep      Keptop
Kepwith  Lex      Lexput   listfile listdirs Params    Pass
```

This is our first River combination. *lextitle* lists titles to the screen, and if we send the output to another module named *columnit*, we get a neater display.

Although this is a very simple example it demonstrates two aspects of River programs.

1. River programs do as little as possible.
2. River programs can easily be combined to create different forms of output as needed by other systems, other applications or people.

If you run

```
lextitle | columnit
```

you will likely be pleasantly surprised at the speed. A great deal of the River speed comes from the smallness of the River programs. *lextitle* is 7K, as is *Columnit*. A 7200 rpm disk will take a millisecond or two to find and load them after which they are running.

Compare this with VBScript, which has a single, main, DLL of 400+ K and it probably loads others. You get similar numbers for PHP and Perl. River modules can often finish a task before environments like VBScript can load.

For running repetitive, looping, scripts River's performance advantage is more extreme. This is because the small River modules will reside in your L1 Cache and not be doing a lot of disk access or disk swaps as do environments like VBScript. Here's a rough chart of the speeds.

<i>Device</i>	<i>Speed Mb/s</i>	<i>Speed Ratio ( to Disk )</i>
Disk	34	1
Memory	1395	41
L2 Cache	4072	120
L1 Cache	16392	482

So if the programs can stay in the L1 cache, they will be about 500 times faster than getting

things from disk, in the L2 cache over a hundred times faster, and in main memory about forty times faster.

If you are administering a server for many users you will achieve similar efficiencies. For programs, small is good, very, very good.

## River Procedure Files

This initial re-release of the River has some system maintenance tools. The River is far more flexible than just systems maintenance. Compared to alternate tools it is very good indeed at systems maintenance.

- 2.bat Directory navigation simplified
- DelTemps deletes temporary files everywhere.
- DelCache empties cache directories
- BigFiles shows you the fat files.
- Lex is an aid to maintaining the lexecon.
- Table2Frm modifies tables that are poorly presented on the web to something more useful.

One procedure is already familiar, a simple one:

```
lextitle | columnit
```

Another we could consider is one to delete all the files from \...\tmp and \...\temp directories. This is a useful thing to do at boot time.

### 2.bat

```
2 drivelist token
```

2 sends you to a new directory.

drivelist may be a list of drives, as in *c;d:...* or *\** for all drives or *.* For the current drive.

*Token* is a token that must exist in the last element of the directory name So to transfer to **c:\windows\internet logs**

one could enter things like:

```
2 c: logs
```

or

```
2 . internet
```

or

```
2 * ernet
```

On Windows NT variants the handling of lists for batch files or command procedures differs from that for programs. On XP, for example, you can not enter a list for a batch file such as:

```
c:,d:
```

You must quote it otherwise Windows will incorrectly convert the comma to a space and call the items separate parameters. Boring of them to err so.

```
"c:,d:"
```

## Tabl2Frm.bat

```
Tabl21frms url no_lines_for_top
```

as in (all one line)

```
tbl2frms  
http://www.cpc.noaa.gov/products/analysis\_monitoring/cdus/degree\_days/hfstwpws.txt 13
```

Tables to Frames changes some poorly prepared web pages such as

[http://www.cpc.noaa.gov/products/analysis\\_monitoring/cdus/degree\\_days/hfstwpws.txt](http://www.cpc.noaa.gov/products/analysis_monitoring/cdus/degree_days/hfstwpws.txt)

to something more useful like

<http://www.lccdigital.com/tbl2frms/main.htm>

The *URL* is the address of a page that is one of those tables.

*no\_lines\_for\_top* is the number of lines in the top after blank lines are removed. This may take a little fiddling for a particular page until you have it right.

The original ruins its usefulness as the title explanation of the columns is lost as one scrolls down. Tabl2frm simply puts the top in a frame, and the bottom, so they can be scrolled independently.

Test.bat is included in the kit. Test.bat shows you the original and the processed versions of the url above.

## DelTemps.bat

DelTemps.bat deletes all the files with an ext or extension of tmp, wrk, and bak. It too is pretty simple in the River:

```
@echo off  
dels "+listfile * *.tmp,*.wrk,*.bak /s"
```

We have fuller and formal definitions for the programs below. For the moment all we need to know is that *Dels* deletes things, and *listfile* lists file names.

The line starts off with *Dels*, so we know we are deleting things. What things/

```
" +listfile * *.tmp,*.wrk,*.bak /s"
```

The "+ ... " convention says:

1. Run the quoted command.
2. Use the output as parameters for the calling command, in this case *Dels*.

We are going to delete all the file listed by

```
listfile * *.tmp,*.wrk,*.bak /s
```

The first asterisk after *listfile* says, "look at all mapped and local drives". Pretty handy, that. If

you only wanted to do C: and D: you could have the *listfile* command be:

```
listfile C:\,D:\ *.tmp,*.wrk,*.bak /s
```

This introduces a parameter, that is, several items separated by commands with no internal blanks (unless quoted). **C:\,D:\** is a command list of two items, obviously.

Another parameter list is the masks to use:

```
listfile C:\,D:\ *.tmp,*.wrk,*.bak /s
```

This command list says find all the files with a suffix of tmp, wrk or bak./

The /s simply says, “and do subdirectories”. So,

```
listfile C:\,D:\ *.tmp,*.wrk,*.bak /s
```

lists all the tmp, wrk and bak files in all subdirectories on C: and D:.

```
dels "+listfile C:\,D:\ *.tmp,*.wrk,*.bak /s"
```

deletes the files, if you have the privileges and they are not locked, or busy.

*Dels* is a simple program. *listfile* is pretty simple, particularly for a programmer who did the very first *FindFile*, submitted to Norton and shipped by him as *FileFind*.

*Dels* and *listfile* are both 11 K.

### DelCache.bat – Delete Cache Files

The *DelCache.bat* procedure empties all the cache directories. This procedure deletes all the files but leaves the directories just so IE7 or some other program will not have vapours if their cache directory is not found. With this in mind we wrote *Dels* so that if the item is a directory name it simply empties the directory and all lower ones. Our command would be simply:

```
dels list of directories to delete
```

How do we create a list of subdirectories? There is a River program called Listdirs You will not be surprised that it lists directories. To empty all the cache files we can enter

```
dels "+listdirs * cache /e"
```

The Listdirs parameters are similar to *listfile*, briefly:

1. Drive parameter list, \* for all drives.
2. A *token* that must be in the last portion of the directory name. Cache directories could be found with a token like *cache*, or *cac* but not *cashew*.
3. /e says make the last part of the directory name be exactly the token entered. With the /e switch the command above will get **C:\stuff\cache** but not **C:\stuff\cacheprograms**. Without the /e switch it would list both.

## Ordinary Pipes

You could use, if you prefer, ordinary pipes. Some people prefer to have the most important command to the left of the line, some prefer ordinary pipes. Shrug.

```
listdirs * cache /e | dels
```

## Non-Cache Cache Directories

We found that some applications actually put data and code in a subdirectory named `...cache`. Not too swift of them. Pretty dumb, actually. Joomla, an otherwise excellent product, did this. So when we empty all the caches we need to skip the one under Joomla.

To skip these we can use a file with tokens of the names to be skipped in them. For `c:\htdocs\joomla\includes\cache` we could have a line in the file which says, for example, "joomla".

Is skipping dirs built into *Dels*, which deletes a list of things? Is skipping dirs built into *Listdirs* which lists directories? Neither. *Delwithi* deletes selected lines from a list. Detailed syntax is below but

```
Delwithi any joomla
```

removes any lines in a list with "joomla" in them.

And,

```
listdirs * cache /e | delwithi any joomla
```

will get a list of directories ending an "cache" with all those in "joomla" removed./

Maybe Joomla is the only application in the world with a mis-named sub-directory, "cache". I doubt it, so we have to be ready for several such.

There are two ways to handle this. One is to add them to the *Delwithi* statement. Suppose two games named GT-500 and AlpineSlide also had directories named cache that were not used for cache files but something else.

```
Delwithi any joomla GT-500 alpineslide
```

would do the job.

Or, an *indirect parameter* could be used. An indirect parameter is one with a preceding @ sign and it replaces itself with the values in the file. The command

```
Delwithi any @skipdirs
```

would eliminate from the River list of directories any with tokens from the file **skipdirs**.

File **skipdirs** would then have the lines:

```
Joomla  
gt-500  
alpineslide
```

and our complete command would be

```
Dels "+listdirs * cache /e | delwithi any @skipdirs"
```

or

```
listdirs * cache /e | delwithi any @skipdirs | dels
```

and here is *DelCache.bat* a little procedure that deletes all files in cache subdirectories except those named in file skipdirs. If there is no **skipdirs** file it shows an explanatory message

```
@echo off
if not exist skipdirs goto noskip

:doit
dels "+listdirs * cache /e| delwithi any @skipdirs"
goto end

:noskip
cls
echo Create a file named 'skipdirs' with tokens for
echo subdirectories to skip.
echo.
echo Run "delcache" again, then, to delete the files in . . .
echo -----
listdirs * cache /e
pause
goto end
:end
```

### BigFiles.bat – List The Biggest Files On The System

From the lexicon, using *usage bigfiles*<enter> you will see:

```
Bigfiles command procedure

Bigfiles [drivelist [masklist [maxlines ] ] ]

Bigfiles lists the biggest masklist files on drivelist up to maxno lines.

Defaults

drivelist *
masklist *.*
maxno 35

If file skipbigs exists, then bigfiles will skip files matching the list using a Delwithi

For drivelist you may use paths, quoted if internal blanks, as in

bigfiles "c:\my documents"

For a production setting you can then skip files for which you can do little such as

pagefile.sys
ntuser
.iso
. . .
```

```
sa: listfiles delwithi insleadb selectab
kw: files size manage
```

and if you run *bigfiles* you will see output, depending on your files such as

```
723,138,560 h:\simplymepis_6.0_i386.iso
715,732,992 f:\livecd-i686-installer-2006.1.iso
419,430,400 h:\pagefile.sys
188,743,680 f:\win386.swp
182,452,224 c:\cdriveback\full.1.dar
179,830,213 e:\devnet\msdn\kb.ivt
123,947,626 c:\cdriveback\full.2.dar
96,964,553 e:\kodak\card\archive.zip
83,036,307 e:\devnet\msdn\vcsample.ivt
52,267,008 h:\down\dsl-3.1.iso
44,102,848 e:\devnet\msdn\prodsamp.ivt
36,260,864 e:\dev7\modemdoc\usrv90v.doc
33,856,153 g:\program files\java\jre1.5.0_07\lib\rt.jar
32,156,353 e:\devnet\msdn\tatech.ivt
29,900,800 h:\down\gparted-livecd-0.3.3-0.iso
28,022,272 h:\programming\lcc\bin\win32.fts
27,130,538 e:\vb5\vbonline\vbonline.m14
26,632,584 h:\down\mysql-4.0.26-win32.zip
26,483,263 h:\mysql\data\terrazin-bin.000013
23,290,672 e:\quask_formartist.exe
23,290,672 e:\p\od\quask_formartist.exe
22,154,833 h:\java\lib\rt.jar
20,033,075 h:\programming\lcc\bin\win32.hlp
19,272,787 e:\devnet\msdn\dmsdn.ivt
19,113,600 g:\down\jre-1_5_0_07-windows-i586-p.exe
19,058,331 h:\down\virtual pc 2004 sp1.zip
19,058,331 c:\dar\roo\sav\virtual pc 2004 sp1.zip
19,058,331 c:\dar\roo\sav\sav\virtual pc 2004 sp1.zip
18,629,120 h:\okdown\mysql_essential_5_0_20_win3.msi
18,597,793 h:\ooo\share\dict\ooo\th_en_us_v2.dat
18,584,143 e:\devnet\msdn\taplat.ivt
18,391,040 c:\dar\roo\sav\sav\perldoc-html.tar
18,391,040 c:\dar\roo\sav\perldoc-html.tar
18,391,040 c:\dar\perldoc-html.tar
17,725,952 e:\msvminst\virtual pc 2004 sp1\microsoft virtual pc 2004
msdn.m
```

Looking at this list I notice that sometime ago I must have dragged a copy of

c:\dar\roo\sav\ to c:\dar\roo\sav\sav\ I'll go in and delete the bad drag-n-drop and save some space.

Bigfiles.bat works and is useful. The parameters, *drivelist*, *masklist* and *maxno* and the optional file *skipbigs* can be used for daily, monthly or other scans of your disks looking for large files that are no longer useful. To check users' file on an NT environment system, where home directories are in **\Documents and Settings**, one might do a command

```
bigfiles "C:\documents and settings" *.* 200
```

with a *skipbigs* files set to ignore the system directories

```
Administrator
All Users
All Users.WINDOWS
```

```
Default User
Default User.WINDOWS
Desktop.ini
Grendel
LocalService
LocalService.NT AUTHORITY
LocalService.NT AUTHORITY.000
NetworkService
NetworkService.NT AUTHORITY
NetworkService.NT AUTHORITY.000
tim oleary
```

Our file would then be:

File skipbigs:

```
service
default
all users
```

## River Application Kits

An *Application Kit* is a zip file containing some batch files and data for a purpose using. They are also good for learning if you work through them.

A one liner about debugging is in order: If you think you have problems with a line such as

```
ifl %1| tabonspc| cvt dropbrak.cvt> log.csv
```

you can easily cut the line to a previous River step to see intermediate results, as in

```
ifl %1| tabonspc
endit
| cvt dropbrak.cvt> log.csv
```

Where *endit* is a one line batch file that does nothing but terminate other batch files and can be created with

```
echo >endit.bat
```

## Apache Access Logs Application Kit

Web servers make access logs showing who asked for what, when. Here is a piece of one:

```
68.122.28.199 - - [19/Feb/2007:00:01:32 -0500] "GET /content/view/65/62/
HTTP/1.1" 200 14326 lccdigital.com
"http://www.stumbleupon.com/refer.php?url=http%3A//lccdigital.com/content/
view/65/62/" "Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1" "-"
68.122.28.199 - - [19/Feb/2007:00:01:33 -0500] "GET
/templates/metropolitan/css/template_css.css HTTP/1.1" 200 9344
www.lccdigital.com "http://lccdigital.com/content/view/65/62/"
"Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1" "-"
68.122.28.199 - - [19/Feb/2007:00:01:35 -0500] "GET
/templates/metropolitan/images/main-back.jpg HTTP/1.1" 200 618
www.lccdigital.com
"http://www.lccdigital.com/templates/metropolitan/css/template_css.css"
"Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1" "-"
```

Ugly stuff.

Everyone has programs to present you with formatted information from the raw data, which is fine, unless

- you need another format or
- you'd just rather do it yourself or
- you have logs with different formats you'd like to combine, a perfect job for the River.

Anyway, download [weblog.zip](#). It has in it some test data, test.log, a daily from 1and1, and some batch files which you can run then dissect and recombine into your own procedures.

## Processing Logs ProcLog 1 To 5

Proclog1.bat, Process Log 1, shows the unique visits:

```
Unique visits 5
```

Proclog2.bat shows the browsers people are using:

```
count  Browser
21     Mozilla/5.0 Macintosh
25     Mozilla/5.0 Windows
23     Mozilla/5.0 Windows
12     Mozilla/5.0 X11
19     Mozilla/5.0 X11
```

Proclog3.bat shows browsers and system OS environment, followed by counts:

```
Mozilla/5.0  Linux i686      en-US  31
Mozilla/5.0  PPC Mac OS X Mach-O  en-US  21
Mozilla/5.0  Windows NT 5.1  en-GB  25
Mozilla/5.0  Windows NT 5.1  en-US  23
```

*Note:* Proclog 3 through Proclog 5 depend on the information the browser or spider returns to the server and there is no standard for this. When used with large logs you will get some odd lines, ones that go not show useful information for things like platform:

```
SurveyBot/2.3 http://help.yahoo.com/help/us/ysearch/slurp nl
```

If you are preparing reports for management you might wish to run a *delwithi* to take out these lines lest they be confused further.

Proclog4.bat shows the languages used and counts:

```
en-GB  25
en-US  75
```

Proclog5.bat shows the various formats that browsers return information in. It is not too bad for test.log, but wait until you try using the full landl.log. There is not a common format and spider bots are the worst.

```
Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.8.1.1)
Gecko/2006120b 4 Firefox/2.0.0.1
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.8.0.9)
Gecko/20061206 Firefox/1.5.0.9
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1
Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Gecko/20060601
```

```
Firefox/2.0.0.1 (Linux Mint)
Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.1) Gecko/20061205
Iceweasel/2.0.0.1 (Debian-2.0.0.1+dfsg-2)
```

## Switching Log Files

If you just run `proclog1.bat` you will use a file named `log.csv` (which you can toss in a spreadsheet). It is a rendition of `test.log`.

If you want to use another file, e.g. `1And1.log`, run `ProcLog1` with the filename as a parameter.

This will create a new `log.csv` file from the log file.

All `proclog?.bat` files behave the same way, without a parameter, use the current `log.csv` file, with a parameter, make a new one. With larger log files this saves some time. For example

```
proclog1 1and1.log
Unique visits 1231

proclog1 test.log
Unique visits 5

proclog1
Unique visits 5
```

## Proclog1 Annotated

```
@echo off

if not a%1==a call makecsv %1

ifl log.csv| selectab 1 4 9 10 11 > log2.csv

rem break the date column into date and time
ifl log2.csv| selectab 2 | snipto : > dates
ifl log2.csv| selectab 2 | snipfrom : > times

ifk log2.csv | selectab 1 3 4 5| addcols dates | addcols times> temp
dels dates times >nul:

rem echo get unique visitors
ifk temp| selectab 1| sort| dupenumb| selectab 2 1| counter| prepline
"Unique visits " $$lin
echo.

goto end

help:
echo run %0 logfile to create a log.csv file

:end
```

```
@echo off
```

```
if not a%1==a call makecsv %1
```

Makecsv.bat makes a log.csv out of a something.log file. It looks like:

```
ifl %1| tabonspc| cvt dropbrak.cvt> log.csv
```

```
ifl log.csv| selectab 1 4 9 10 11 > log2.csv
```

With this selectab we choose columns 1, 4. ... which are IP Address, date and time, Request info, and browser info.

```
rem break the date column into date and time
ifl log2.csv| selectab 2 | snipto : > dates
ifl log2.csv| selectab 2 | snipfrom : > times
```

these snips break out the dates and times into files of one column.

```
ifk log2.csv | selectab 1 3 4 5| addcols dates | addcols times> temp
dels dates times >nul:
```

Addcols tacks on a column to the end of the River, so with this we selectab to eliminate the old date/time column and then use addcols to add on new date and time columns

Actually for counting unique visitors all we need is the IP column as our files have only one date. We left in the other information in case you want to make a more comprehensive unique visitor report.

```
rem get unique visitors
ifk temp| selectab 1| sort| dupenumb| selectab 2 1| counter|
prepline "Unique visits " $$lin
```

This ifk single line inputs and kills file temp, selects the IP column, sorts it then numbers duplicate lines. The “selectab 2 1” simply puts the counts in front of the ip addresses.

```
echo.
goto end

help:
echo run %0 logfilefilename to create a log.csv file
:end
```

the above is ordinary cleanup and a usage line in case there is no log.csv file.

Here are some alternatives to give you a view of how different River approaches would work.

## ProcLog1 – Only IP

If we really only wanted a count of unique visits this could be the one line in bold, below.

```
@echo off
if not a%1==a call makecsv %1
ifl log.csv| selectab 1| sort| dupelim | counter| preline "Unique visits "
$$lin

echo.
goto end

help:
echo run %0 logfilefilename to create a log.csv file
:end
```

## Or, Straight From The Log File

For something as simple as counting IP visits we don't even need to make the CSV file as the log file starts with *ipaddress[blank]* and so

```
ifl test.log| snipto " "| sort | dupelim | counter
```

works, too.

## Other ProcLog Batch Files

The other Proclog batch files are very similar. If there is something that is hard to understand in them pleas let us know at [riverdoc@lccdigital.com](mailto:riverdoc@lccdigital.com). Thanks.

## Miscellany

### XP Command Screens

When using command prompt screens in XP, at least some, the screen often changes itself from a full screen display to a smaller, windowed display without user commands to do so. This is called a bug, and if you like full screen displays it is extremely annoying.

*MaxScr* is a little utility to blow the screen back to full size. It makes this Windows bug a little less annoying.

### Internal Line Lengths

We allow River lines of 4096 bytes.

Certain NOTE functions allow for line lengths of 16384 bytes. These limitations may be changed if you wish.

### Parameter Counts And Processing

In general we have allowed for 20 parameters. The River allows *parameter processing* for modules that would benefit and then the limitation is 5,000 parameters in some cases, open ended in others. Edit *Params.inc* and re-compile if you need more than 5,000 parameters.

### Indirect Parameters

A parameter with an initial character of an '@' is an indirect parameter. An indirect parameter says "Use what follows as a file name and find the information there, one item per line."

### Function Lists

You may use a function (program) to define a list by surrounding the function in quotes with a leading ampersand. An example of this might use the River *dels* command, a smarter file deleter:

```
dels "&dir *.bat /s"
```

which runs the command "dir \*.bat /s, and uses the output from the command as a list of items to delete.

You may use compound commands in a function list, as in

```
dels "&dir *.bat /s | delwithi any mybat.bat"
```

### River Size Capabilities

Although this may depend on your system, cache, memory, ... in testing we ran 100 Megabyte files through River without a hitch.

## The Lexecon

The lexecon is an ASCII file with all the modules short description and parameter descriptions. Some use it as a short documentation file. The format rules are simple:

1. A segment has a title line which is the first non-blank line after a “//” line or the beginning of the file.
2. A line with only two slashes, solidi, “//” terminates a segment.

And there are some developing conventions not yet fully implemented:

1. A line beginning with “sa:” has space separated items to *see also*.
2. A line beginning with “kw:” has keywords for the segment.

*Lex* is a command procedure that extracts a segment, lets you edit it, and then replaces it. If you use *lex* you will probably need to replace the line beginning with “z” unless you have the *Z* editor distributed by Harry Newton awhile ago.

## Release 2.1 River Modules List

The River has gone from a few utilities in VBDOS a long time ago, to implementations in C, some in VB5 and back to C again. VBDOS was OK for the 80s, C is slow to code, and VB5 and later are obese mothers that insist on shipping a meg or so for a program that says:

Hello, world.

Now we use BCX as it is a BASIC™ like language and easier for many people to read and alter as needed. It also creates, in conjunction with the LCC32 compiler, acceptably small programs, a few K or so.

We are in the process of converting an older version to this language and the full list of older River modules is in a later chapter.

The Modules in this initial BCX version are, as of March 10, 2007:

addcols	dupelim	fsetsize	osver
cdd	dupenumb	h2adate	parse
choositg	endit	h2adatef	prepend
clip2fil	exist	ifk	prepline
clip2riv	fil2clip	ifl	postpend
columnit	fgetattr	insleadb	retcd
counter	fgetcrea	keep	riv2clip
count	fgetlast	kepbot	selectab
dels	fgetmd5	keptop	setcwd
deltop	fgetmod	kepwith	shortnam
delwith	fgetsha1	kepwithi	snip
delwithi	fgetsize	lextitle	snipfrom
doit	filetail	lexget	snipto
dropdriv	fsetattr	lexput	tabonspc
dropext	fsetcrea	listdirs	toglscr
dropfile	fsetlast	listfile	usage
droppath	fsetmod	maxscr	

For each of the programs, not the procedures, to see details and examples you may run them with a */?* parameter or enter

<i>usage name</i>
-------------------

.Procedure or scrip files are documented in the Lexecon, so usage will work.

### Addcols – Add Columns To The River

```
... | addcols filename | ...
```

Add columns to the end of each line in the River.

The number of lines in the River ought to, must, equal the number of lines in the file. AddCols is mostly used for extracting a column, changing it and adding back in to the River the changed results.

### CDD – Change Directory Down

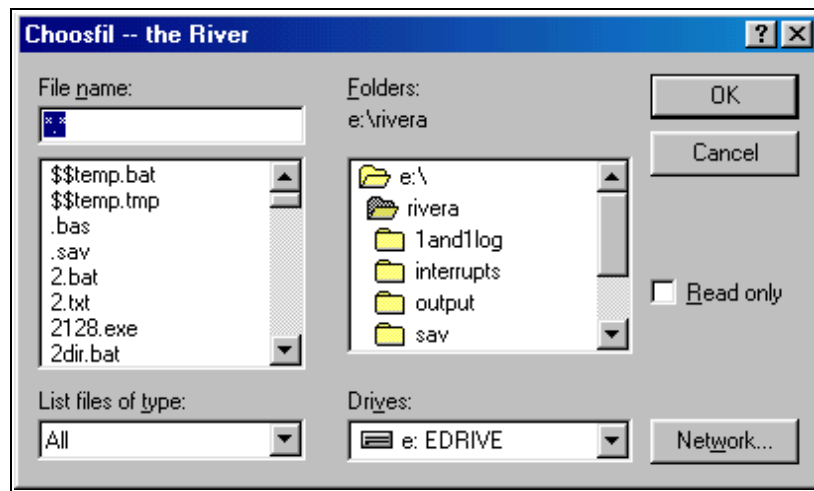
```
Cdd token
```

cdd changed directory down one to the first sub-directory it encounters that has *token* in the last segment of the path.

### Choosfil – User File Selection

```
Choosfil [d:][\path\][filemask] | , , ,
```

Choosfil launches an ordinary Win file screen and places the results in the River:



If the user selects several files, the files are given a line each in the output.

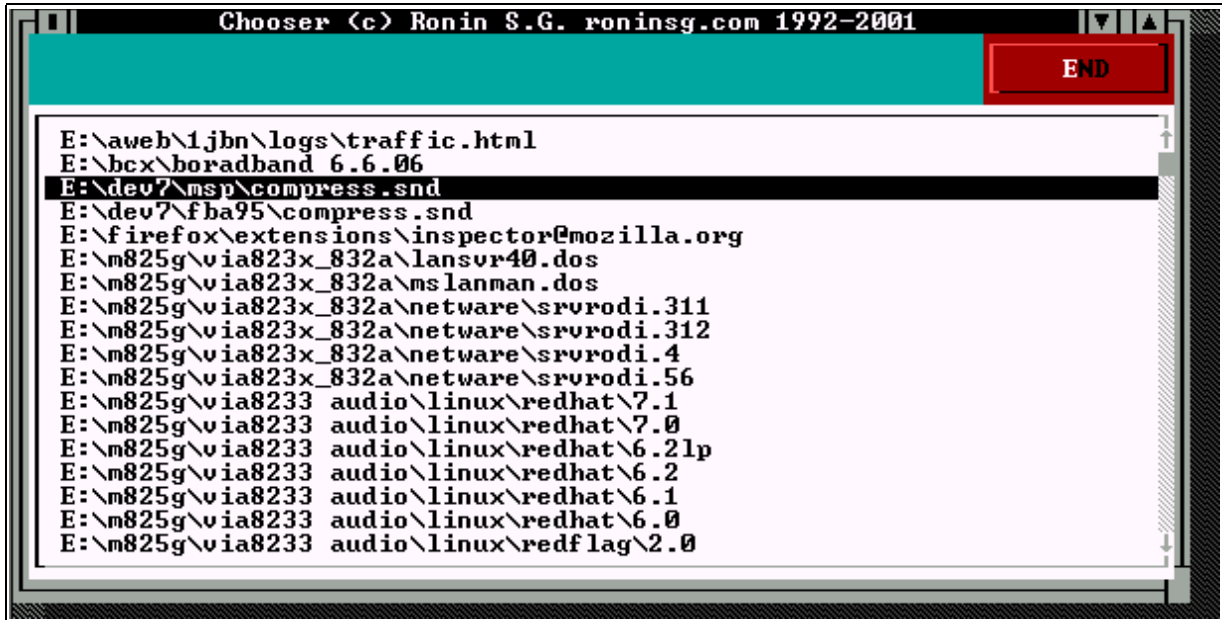
An optional parameter, [d:][\path\][filemask], may have any of the portions but are not to be space separated.

*Filemask*, if found, is an ordinary wildcard file mask.

Optional d: and \path\ are used as ordinarily.

### Choositg – Select An Item From The River

Choositg displays the River in a list for user selection:



... | choositg | ...

### Clip2Fil & Clip2Riv

Send the clipboard to a file or the River.

Clip2fil [d:\[\path\]filename.ext

Clip2Riv | . . .

### Count And Counter

These count the lines in the River. *Count* passes along the lines followed by a count to the next step in the River and *Counter* only outputs a count.

... | count | ...

... | counter

### Columnit – Place The River In Columns For Display Or Reports

river ... | columnit

Columnit is a simple columnizer that arranges River output into columns. Columnit expects short lines and it is not a general purpose columnizer, but it is handy.

### Delwith And Delwithi

... | delwith { any | all | token token ... | ...

```
... | delwithi { any | all | token token ... | ...
```

These delete lines from the River. Delwithi is case insensitive.

Both have the same parameters, a leading *any* or *all* which refers to the type of test with the tokens, followed by tokens.

```
Delwith { any | all } token1 token2 . . .
```

will delete lines with any of the tokens, or all of the tokens. Delwith and Delwithi allow indirect parameters, as in

```
Delwith any @skiplist
```

### Doit

```
... | doit
```

Doit may terminate a River sequence and it will treat each line as a command and run it.

### Dropdriv, Dropext, Dropfile And Droppath

```
... | dropdriv | ...
```

These four are used to remove some of a complete, or canonical, file name, as in

```
c:\tools\batch\test.bat
```

Dropdriv will remove the drive

```
\tools\batch\test.bat
```

Dropext will remove the ext

```
c:\tools\batch\test
```

DropFile will remove the file name

```
c:\tools\batch\
```

DropPath will remove the path information

```
test.bat
```

### Dupelim Eliminates Duplicate Lines

```
... | dupelim | ...
```

*Dupelim* will remove any line that is identical to the one above it. Dupelim assumes you have run a sort beforehand.

### Dupenumb Numbers Duplicate Lines

```
... | dupenumb | ...
```

*Dupenumb* will count all duplicate lines and leave one line followed by a tab, then the count of occurrences of that line.

## Endit – Terminate A Batch File

Endit.bat is a null file that can be used in any batch file to terminate the execution now. Endit is often used when debugging River procedures after splitting a line to see intermediate results. As in changing:

```
ifl test.log| snipto " "| sort | dupelim | counter
```

to, in order to see intermediate results but without losing the rest of the line

```
ifl test.log| snipto " "  
endit  
rem | sort | dupelim | counter
```

## Exist Filename Command

```
Exist [d:\][\path\]filename command
```

If *filename* exists the command is run.

## Fgetattr, Fgetcrea, Fgetlast, Fgetmod, Fgetsize

These routines add an attribute list, creation date, modification date, last access date or size to a river whose first column is a file name, as in

```
dir /b| fgetcrea | fgetlast
```

You may also use

```
FgetAttr [d:\][\path\]filename
```

and similarly for the other four functions

## Fgetmd5, Fgetsha1

These routines add an MD5 or SHA-1 signature to the a river whose first column is a file name, as in

```
dir /b| fgetsha1
```

You may also use

```
FgetMD5 [d:\][\path\]filename
```

and similarly for the SHA-1.

## Fil2Clip – Send The Clipboard T A File

```
Fil2clip [d:\][\path\]filename.ext
```

## Filetail Keeps The Back End Of A File's Data

```
filetail nnn[ k | m] file file ...
```

Filetail throws away the the beginning of a file and keeps the last part.

The first parameter is the new size with an optional “k” for kilobytes or “m” for megabytes.

Filetail can use a River of filenames for input, as in:

```
listfile * *.log /s | filetail 20K
```

Some will prefer keeping the last *n* lines for which

```
ifl filename| kepbot n > temp
del filename
rename temp filename
```

will do adequately.

### Fsetattr, Fsetcrea, Fsetlast, Fsetmod, And Fsetsize

```
River of file names... | Fsetattr attr
River of file names... | Fsetcrea Date
River of file names... | Fsetlast Date
River of file names... | Fsetmod Date
River of file names... | Fsetsize new_size
```

These five programs will set a file's attribute, creation date, last accessed date, modification date or resize the file, respectively.

*Date* is in the form of year(sep)month(sep)day(sep)hour24(sep)minute(sep)sec(sep)milli where (sep) is any of { : | . | (space) | / | - } where year is two or four digits, 1980 being the earliest twentieth C. year, and hour24 is 0 to 23. 1998/3/22 23:59:59 is a legitimate date as is 05.03.12 3:12:14.040 Missing values receive the value zero.

*New\_size* may have a “k” or “m” suffix for kilobytes and megabytes, respectively.

All use the River for the file name(s) and the command line for the new value.

### H2adate H2adatef – Hex To Alpha Dates

```
River of file names ... | h2adate [colno]
River of file names ... | h2adatef [colno]
```

Colno is the column of the hex date, the default is 2.

The file dates start off as hex, Unix style dates, as in this retrieval for the create date from file testf:

```
echo testf | fgetcrea
testf1C75823 91D7FF20
```

These are great for sorting and someone, somewhere is going to write an application relying on hex dates, probably something in forensics, but otherwise this format is awkward. H2adate converts the hex date to the local date format. H2adate works with all of the date retrieval programs.

```
echo testf | fgetcrea | hs2date
testf2/24/07 9:53:39 AM
```

H2adateF is the full version which includes milliseconds and uses the UTC standard time, as in

```
echo testf | fgetcrea | hs2datef
testf2007.02.24 14:53:39.730
```

### Ifk – Insert File And Kill It

```
Ifk filename | River ...
```

Ifk inserts a file just as the DOS command “Type” does. Ifk deletes the file after inserting it to the River.

### Ifl – Insert File

```
Ifl filename | River ...
```

Ifl inserts a file just as the DOS command “Type” does. Ifl is written to accept lines up to 4097 and there is no documentation about Type's reaction to longer lines than it expects.

### Insleadb – Insert Leading Blanks

```
River ... | insleadb colno number
```

Insleadb allows you to right justify a line which can be very important for visual appreciation or sorting.

*Colno* is the column for the operation, and *number* is the number of spaces that the column should have after the operation.

### Keep – Keep Lines In The River

```
Keep token
```

Keep examines each line and if the token is not in the line, the line is eliminated.

Keep is case insensitive, for more precise control see KeepWith and KeepWithI

### Kepbot & Keptop -- Keep The Bottom Or Top Of The River

```
Kepbot n  
Keptop n
```

*n* is the number of lines from the top or bottom to keep, the rest are eliminated.

### Kepwith & Kepwithi

```
Kepwith { any | all } token token token ...  
Kepwithi { any | all } token token token ...
```

*Kepwith* keeps lines in the River which have any of the tokens, if the first parameter is *any* or all of the tokens if you specify *all*.

*Kepwithi* is the same as *kepwith* but case insensitive.

### Lex – Edit A Lexecon Item

```
Lex item_name
```

Lex extracts and item from the lexecon file, allows you to edit it, and then replaces the item.

Lex is a batch procedure that will need you to change the “z” at the beginning of the thirteenth line to your editor instead of the Z editor.

### Lexget & Lexput

```
Lexget item_name  
lexput item_name
```

Lexget gets a lexecon item's text, and lexput puts it back. Lex Put puts it back in case insensitive sort order.

They are used in lex in a fashion not dissimilar to:

```
Lexget ifl
```

```
edit ifl.txt
lexput ifl
```

### Lextitle – Show The Items In The Lexecon File

```
lextitle
```

lextitle shows the titles of the items in the lexecon

### Listdirs – List Directories Like...

```
Listdirs [ {drive: | * | pathlist} [tokenlist [ /e ] ] ] | ... River
```

Listdirs lists directories that have something in the token list inside their last name section, after the penultimate reverse solidus, backslash, ”\”.

### Listfile – List Files Like

```
Listfile [ {drive: | * | pathlist} [masklist [ /s ] ] ] | ... River
```

Listfile lists the files on the drive or in the path list that are matched by any of the masks in the masklist and will search lower directories also if the /s switch is specified.

Examples:

```
listfile * .c, .h /s
```

```
listfile "c:\my documents, g:\" .bas, .inc /s
```

```
listfile @pathlist @masklist /s
```

### Maxscr – Maximize The Screen In A Command Window

```
maxscr
```

Windows sometimes spontaneously decided to make your full screen command prompt window smaller. An annoying bug in Windows if you like to work in full screen mode.

Maxscr re-maximizes it.

### Prepend -- Introduce A File To The Beginning Of The River

```
River ... | postpend [d:][\path\]filename.ext
```

### Prepline – Prepare River Lines

```
River ... | prepline text... "quoted text "... $$lin ...
```

prepline takes each River line and lets you add text to different places. \$\$lin is a special parameter that means “insert the line here”. Text may be inserted many times and \$\$lin may be used many times.

The following will rip all DOC files from c: to dL

The first line creates all directories on D: that are directories on C:, which we need as we will copy to them and the Window copy does not allow copying to a non-existent directory.

The second line lists all the DOC files and then creates a copy command.

```
Listdirs c:\ | droppath | prepline "mkdir d:" $$Lin "> nul:| doit  
Listfile c: *.doc /s |droppath |prepline "copy c:" $$lin " d:" $$lin |doit
```

Notes:

1. To create a list of the directories on C: that have doc files would
  1. List all the doc files on C:
  2. Drop the drive and filename
  3. Sort the results
  4. Eliminate dupes
  5. Now we have a list of all directories needed.
  6. A simple prepline can create a mkdir command to create them on D:

Postpend -- Add A File To The End Of The River

```
River ... | postpend [d:][\path\]filename.ext
```

Retcd – Show The Return Code For A Program

Retcd shows the return or exit code or condition for a program.

```
Retcd fc file2 file2
```

Will show the return code for the file compare.

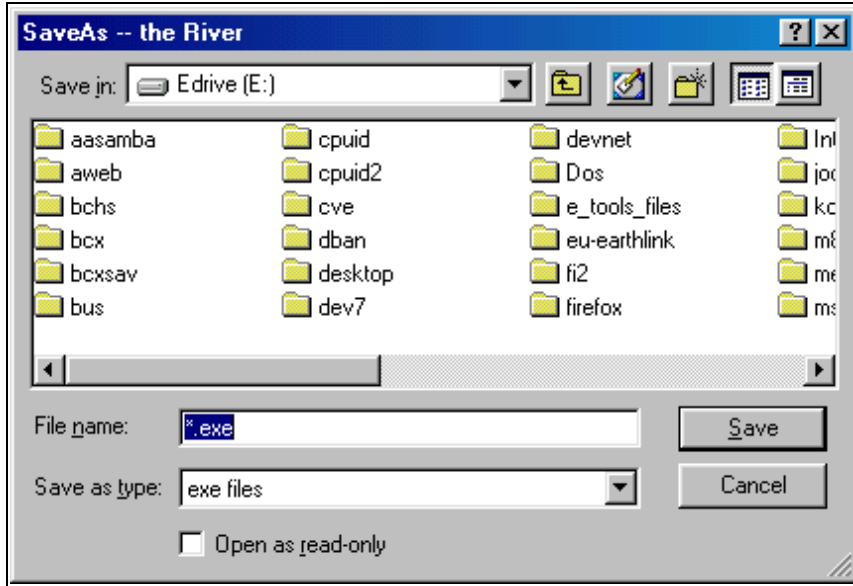
Riv2clip – Send The River To The Clipboard

```
. . . | riv2clip
```

### Saveas – Save The River To A User Selected File Name

```
Saveas [d:] [\path\ [filemask.exemask]
```

Saveas brings up an ordinary Win save as box and lets the user select the file to which to save the River



### Selectab – Select Tabbed Columns

```
Selectab col1 col2 col3 ... col20
```

The River is generally tab separated columns. If the columns are not in the order you wish you may run selectab to move columns, and delete or replicate columns.

The first parameter is the number of the incoming column that will be the new first column;The second parameter is the number of the incoming column that will be the new second column; and so forth.

If the incoming information is

```
a    b    c    d
```

and the selectab parameters are:

```
selectab 3 1 2 3
```

then the output will be:

```
c    a    b    a
```

### Setcwd – Set The Current Working Directory

SetCWD uses both a drive and path name and sets the CWD.

```
Setcwd d:\path\
```

### Shortnam -- Convert File Or Path Names To 8.3

```
. . . | shortnam | . . .
```

Shortnam reduces long file and path names to short, 8.3 format, names.

Warning: On XP certain system directories are not converted using the Win32 API GetShortPathName. An example is any \Documents and Settings\username\Application Data The “ Documents and Settings “ is converted correctly but “Application Data” is not.

The bug has been reported to Microsoft.

### Snip – Take Segments Out Of Lines

```
... | snip snipspec1 [ snipspec2 ... snipspec9] | ...
```

where a *snipspec* is one of

- s-f where s is the start character number and f is the finish character number
- \*-f from the beginning, character 1, to the f<sup>th</sup> character
- s-\* from character s to the end
- /t insert a tab character

### Snipfrom – Take A Segment Starting With A String

```
... | snipform string | . . .
```

where *string* is a sequence of characters that will not be included and that start the snip.

```
Echo abcde| snipfrom c
```

would result in *de*.

### Snipto – Take A Segment Up To A String

```
... | snipto string | . . .
```

where *string* is a sequence of characters that will not be included and that end the snip.

```
Echo abcde| snipto c
```

would result in *ab*.

### Tabonspc

```
... | tabonspc | ...
```

tabonspc inserts tab characters where there are spaces. If there are many spaces together, on tab is inserted.

Tabonspc is often used to start the conversion of text line output to useful columns/

### ToglsCr – Toggle Full Screen Mode For Cmd Windows

```
ToglsCr
```

ToglsCr swaps a DOS box or command window from full screen to windowed and back again every three seconds. Hit any key to exit ToglsCr when the screen is the size you want it.

XP spontaneously goes from full screen display to windowed with user command sometimes. ToglsCr will put it back. Also, if you use windowed mode usually and wind up in Full screen there is no easy way, other than toglsCr, to get back.

## Usage – Quick Documentation

```
Usage section_name [lexecon_filename]
```

Usage will display the section of section name from the lexecon file.

*lexecon\_filename* defaults to “lexecon” and if it is not in the current working directory a path search is made for it.

## The Lexecon™

“Lexicon” would be harder to prevent abuse buy trademarking.

### Lexecon Format

A lexecon file is an ASCII text file with lines no more than 4096 bytes, but no more than 80 is recommended.

A *Section*, sometimes called an Item, is a series of lines starting after either the top of the file or a end of section marker.

A *End of Section Marker* is a line with only two slash characters, solidi, “//:”

The *Title* or *Key* to the section is the first non-blank line in the section.

### The Count And Counter Lexecon Section

```
Count Counter
... | count | ...
... | counter

Count and Counter count the lines in the river.

They also set an return code of the number of lines, over 254 is
returned
as 254.

Count passes along the River, and Counter terminated the River.

... | Count | ...
... | Counter

Count prints the count unless it is given /s, silent, as a
parameter:
... | Count /s | ...

In this case it outputs the count to stderr, that is it shows on the
screen but does not go to the River.

//
```

## **Windows Problems and Warnings**

There are many versions of Windows, heck, there are many different versions of Windows XP.

Many of them have different bugs than others. We are not funded (you can help with this) to keep track of all the Windows bugs on all the Windows variants.

Instead, when we notice a Windows bug we equip you with the code to test your version and take preventive actions as needed.

## Warning on Parameters for Batch Files

### Windows Errs On Some Versions

When some parameters are passed to programs they differ from the same parameters passed to a batch file or command procedure.

An invocation like

```
program a:,b:
```

will be passed correctly to a program as

```
a:,b:
```

however if you use the same parameter for invoking a command procedure t, only on some Windows, the comma is incorrectly converted to a space and the single parameter becomes two parameters.

```
Proc a:,b:
```

results in the %1 parameter as “a:” and the %2 parameter as “b:”

This could, and probably does, wreak havoc with moving command procedures to other systems and also with creating command procedure wrappers for programs.

### Work Around

There is a workaround, quoting the string that has commas forces them to be transferred correctly.

You may check your system with a simple batch file:

testparms.bat

```
echo Parameter 1 : %1  
echo Parameter 2 : %2
```

and run it with an entry such as

```
testparms a:,b:
```

## NUL: Windows discrepancy

### NUL: Vs NULL:

On some versions of Windows you can use the NUL: or NULL: pseudo device to eat the output of a command.

Command outputs are usually internally known as stdout, standard output, and stderr, standard error.

On some systems NUL: not only deletes all the stdout information but also the stderr information.

You may test this easily with the count program, which, in silent mode uses stderr for the count display.

#### Testnul.bat

```
@echo off
echo Test NUL:
Dir | count /s >NUL:

echo.
echo -----
echo Test NULL:
Dir | count /s >NULL:
```

On some systems only the second count will show

### Work Around

So far only NUL: has the problem on only some systems.

If you restrict yourself to only using the NULL: terminator there do not appear to be problems.