ECHO2CON

```
Syntax:     ECHO2CON
```

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter sends the data from its standard input to both the standard output and the standard error device. REDIRECTION of the standard input and standard output is possible. The standard error device is always the console.

ECHO2CON is handy when you are troubleshooting a filter sequence. Placing ECHO2CON inside the sequence allows you to see intermediate results without disrupting the sequence itself.

Example using ECHO2CON, LOWER, UNIQUE and WORDS
ECHO2CON ERRORLEVELs

Example using ECHO2CON, LOWER, UNIQUE and WORDS:

```
LOWER < book.txt | WORDS '- | ECHO2CON | SORT | UNIQUE >words.txt
```

will create WORDS.TXT; a LOWER-case list of all the UNIQUE WORDS contained in BOOK.TXT. ECHO2CON will display those WORDS before they are SORTed.

ECHO2CON ERRORLEVELs

```
ERRORLEVELs:
  None.
```

AFTER

```
Syntax:     AFTER [+n|-n|+0n|-0n][key|ch[ar]] [<prefix>]
```

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This parse filter only includes the remainder of each line AFTER the **KEY** character. The beginning of the line is excluded. The **KEY** itself is excluded. The default **KEY** is groups of spaces and tabs. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

+n only includes the remainder AFTER the Nth occurrence of the **KEY**, counting forward from the beginning of each line. If the Nth **KEY** does not exist, includes the entire line as-is. Default is +1.

-n only includes the remainder AFTER the Nth occurrence of the **KEY**, counting backward from the end of each line. If the Nth **KEY** does not exist, includes the entire line as-is.

+0n only includes the remainder AFTER the Nth occurrence of the **KEY**, counting forward from the beginning of each line. If the Nth **KEY** does not exist, the entire line is excluded. N defaults to 1.

-0n only includes the remainder AFTER the Nth occurrence of the **KEY**, counting backward from the end of each line. If the Nth **KEY** does not exist, the entire line is excluded. N defaults to 1.

KEY is optionally any single, visible, (Graphic black-space) character. Letters are case-sensitive.

CHAR excludes +N CHARacters counting forward from the beginning or includes -N CHARacters counting backward from the end of each line, instead of using a **KEY** character. Tabs are treated as one CHARacter. Default is +1.

HINT: If your **KEY** is a digit, place it before your + or - option on the command line; e.g. 2+3, not +32, not +3 2, not +3 2. Similarly, the **KEY** may be + or -.

PREFIX is an optional string that replaces the excluded portion at the beginning of each line. BATCH replaceable parameters and the following escape sequences are supported within the PREFIX:

//	/
/nnn	any single decimal byte. nnn may range from 0 to 255.
/b	Backspace, (BS: /8) does not delete the previous byte.
/f	Form-Feed, (FF: /12)
/n	liNe-feed, (LF: /13)
/r	carriage-Return, (CR: /10)
/s	space, (/32)
/t	Tab, (HT: /9)
/q	" (Quote: /34).
/v	| (Vertical tab: /11).
/:	% (percent: /37).
/[	< (less-than/left angle bracket: /60).
/]	> (greater-than/right angle bracket: /62).
/d0	inserts the text BEFORE and including the KEY.
/d1	inserts another copy of the text AFTER the KEY.
/d2	inserts another EoL as found in the standard input.
/d3	inserts as /d0/d1/d2 above.
/#	inserts the number of this line.
/&	inserts the number of bytes examined.

If **/** is followed by any other character, it is interpreted according to SR.

Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE

Example using AFTER, BEFORE and UNIQUE
Example using AFTER, BEFORE, PREFIX and SUFFIX
Example using AFTER, BEFORE, UNIQUE, UPPER and WORDS
AFTER ERRORLEVELs

Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE:

```
BEFORE +3 <p.s|SINGLES :|LOWER|find ":call:"|AFTER -1: |sort|UNIQUE /U
```

will FIND all CALLs in assembly file P.S, delete their comments (BEFORE +3), ignore tabs (SINGLES) and differences in case (LOWER), extract the labels and "CALL" (AFTER -1:), SORT them and display those which are called only once (UNIQUE /U). This reveals subroutines that could be integrated into the calling section. It still works if there is neither space nor tab between "label:" and "call".

AFTER ERRORLEVELs

Errors are per SR ERRORLEVELS.

BEFORE

Syntax: BEFORE [+n|-n|+0n|-0n][key|ch[ar]] [<suffix>]

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This parse filter only includes the beginning of each line BEFORE the **KEY** character. The remainder of the line is excluded. The **KEY** itself is excluded. The default **KEY** is groups of spaces and tabs. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

+n only includes the beginning BEFORE the Nth occurrence of the **KEY**, counting forward from the beginning of each line. If the Nth **KEY** does not exist, includes the entire line as-is. Default is +1.

-n only includes the beginning BEFORE the Nth occurrence of the **KEY**, counting backward from the end of each line. If the Nth **KEY** does not exist, includes the entire line as-is.

+0n only includes the beginning BEFORE the Nth occurrence of the **KEY**, counting forward from the beginning of each line. If the Nth **KEY** does not exist, the entire line is excluded. N defaults to 1.

-0n only includes the beginning BEFORE the Nth occurrence of the **KEY**, counting backward from the end of each line. If the Nth **KEY** does not exist, the entire line is excluded. N defaults to 1.

KEY is optionally any single, visible, (Graphic black-space) character. Letters are case-sensitive.

CHAR includes +N CHARacters counting forward from the beginning or excludes -N CHARacters counting backward from the end of each line, instead of using a **KEY** character. Tabs are treated as one CHARacter. Default is +1.

HINT: If your **KEY** is a digit, place it before your + or - option on the command line; e.g. 2+3, not +32, not +3 2, not +3 2. Similarly, the **KEY** may be + or -.

SUFFIX is an optional string that replaces the excluded portion at the end of each line. BATCH replaceable parameters and the following escape sequences are supported within the SUFFIX:

//	/
/nnn	any single decimal byte. nnn may range from 0 to 255.
/b	Backspace, (BS: /8) does not delete the previous byte.
/f	Form-Feed, (FF: /12)
/n	liNe-feed, (LF: /13)
/r	carriage-Return, (CR: /10)
/s	space, (/32)
/t	Tab, (HT: /9)
/q	" (Quote: /34).
/v	| (Vertical tab: /11).
/:	% (percent: /37).
/[	< (less-than/left angle bracket: /60).
/]	> (greater-than/right angle bracket: /62).
/d0	inserts another copy of the text BEFORE the KEY.
/d1	inserts the text including and AFTER the KEY.
/d2	inserts another EoL as found in the standard input.
/d3	inserts as /d0/d1/d2 above.
/#	inserts the number of this line.
/&	inserts the number of bytes examined.

If **/** is followed by any other character, it is interpreted according to SR.

Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE
Example using AFTER, BEFORE and UNIQUE
Example using AFTER, BEFORE, PREFIX and SUFFIX
Example using AFTER, BEFORE, UNIQUE, UPPER and WORDS
BEFORE ERRORLEVELs

Example using AFTER, BEFORE and UNIQUE:

```
DIR /a-d c:\ |AFTER +4 "CALL john " |BEFORE "-1." ".*" |UNIQUE |find ".*"
>tmp.cmd
```

creates a working COMMAND file called TMP.CMD. TMP.CMD will include a line for each file in C:\ with an extension, excluding directories. For the file "C:\setup.log", the resulting line would look like this:

```
    CALL john setup.*
```

Executing TMP.CMD requires that JOHN.CMD exists, too. The code of JOHN.CMD would use %1 to do something with the named file(s).

AFTER
BEFORE
UNIQUE

BEFORE ERRORLEVELs

Errors are per SR ERRORLEVELS.

LOWER

Syntax: LOWER

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter translates every letter to LOWER-case. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the sizes of words are only limited by memory.

Example using LOWER, UNIQUE and WORDS
Example using ECHO2CON, LOWER, UNIQUE and WORDS
Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE
Example using LOWER and SINGLES
LOWER ERRORLEVELs

Example using LOWER, UNIQUE and WORDS:

LOWER < book.txt | WORDS '- | sort | UNIQUE /C | sort /R > words.lst

will create and fill WORDS.LST with a LOWER-case list of all the UNIQUE WORDS contained in BOOK.TXT, including and SORTed according to their frequencies. Furthermore, the most frequent words will be listed first and words with equal frequencies will be listed in Reverse alphabetical order. The list provides a statistical glossary of BOOK.TXT.

LOWER ERRORLEVELs

Errors are per SR ERRORLEVELS.

PREFIX

Syntax: PREFIX <string>

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter prepends **STRING** to the beginning of each line. The **STRING** should be surrounded by double quotes if it includes < |, >. Double quotes may also be a part of the **STRING**. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

BATCH replaceable parameters and the following escape sequences are also supported within the **STRING**:

//	/
/nnn	any single decimal byte. nnn may range from 0 to 255.
/b	Backspace, (BS: /8) does not delete the previous byte.
/f	Form-Feed, (FF: /12)
/n	liNe-feed, (LF: /13)
/r	carriage-Return, (CR: /10)
/s	space, (/32)
/t	Tab, (HT: /9)
/q	" (Quote: /34).
/v	| (Vertical tab: /11).
/:	% (percent: /37).
/[	< (less-than/left angle bracket: /60).
/]	> (greater-than/right angle bracket: /62).
/d1	inserts another EoL as found in the standard input.
/#	inserts the number of this line.
/&	inserts the number of input bytes examined.

If **/** is followed by any other character, it is interpreted according to SR.

Example using PREFIX and SUFFIX
Example using AFTER, BEFORE, PREFIX and SUFFIX
PREFIX ERRORLEVELs

Example using PREFIX and SUFFIX:

DIR /a-d/b | PREFIX "MOVE " | SUFFIX " %temp%" > tmp.bat

creates a working BATCH file called TMP.BAT. That file will include a line for each file found whose archive attribute is set. Directories will be excluded. With the "TEMP" ENVIRONMENT variable SET to "c:\junk", the resulting line in TMP.BAT for the file AUTOEXEC.BAT would look like this:

```
    MOVE AUTOEXEC.BAT C:\JUNK
```

Executing TMP.BAT would move all files whose archive attribute is set to C:\JUNK.

PREFIX
SUFFIX

PREFIX ERRORLEVELs

Errors are per SR ERRORLEVELS.

SINGLE

Syntax: SINGLE

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter removes all blank lines. The input's original EoL style is maintained. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

Example using SINGLE and SUFFIX
SINGLE ERRORLEVELs

Example using SINGLE and SUFFIX:

```
SINGLE < abc.txt | SUFFIX /d1 > prn:
```

Prints ABC.TXT with consistent double-line spacing.

SINGLE
SUFFIX

SINGLE ERRORLEVELs

Errors are per SR ERRORLEVELS.

SINGLES

Syntax: SINGLES [string]

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter reduces sets of one or more spaces or tabs to a single space. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the sizes of white-spaces are only limited by memory.

STRING is used as the replacement for sets of one or more spaces or tabs instead of replacing them with one space. **STRING** may contain anything, including BATCH replaceable parameters, but some characters require quotes as follows:

```
"<"
"|"
"'"
">"
" (tabs) "
" (spaces) "
```

Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE
Example using LOWER and SINGLES
SINGLES ERRORLEVELs

Example using LOWER and SINGLES:

SINGLES <program.asm | LOWER | find " proc "

will reliably FIND all procedures named in assembly file PROGRAM.ASM, ignoring case, tabs and words like "Proceed" and "Microprocessor", producing a table of procedures contents.

LOWER
SINGLES

SINGLES ERRORLEVELs

Errors are per SR ERRORLEVELS.

SUFFIX

Syntax: SUFFIX <string>

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter appends **STRING** to the end of each line. The **STRING** should be surrounded by double quotes if it contains spaces, tabs, |, < or >. Double quotes may still be a part of the **STRING.** REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

BATCH replaceable parameters and the following escape sequences are also supported within the **STRING**:

//	/
/nnn	any single decimal byte. nnn may range from 0 to 255.
/b	Backspace, (BS: /8) does not delete the previous byte.
/f	Form-Feed, (FF: /12)
/n	liNe-feed, (LF: /13)
/r	carriage-Return, (CR: /10)
/s	space, (/32)
/t	Tab, (HT: /9)
/q	" (Quote: /34).
/v	| (Vertical tab: /11).
/:	% (percent: /37).
/[	< (less-than/left angle bracket: /60).
/]	> (greater-than/right angle bracket: /62).
/d1	inserts another EoL as found in the standard input.
/#	inserts the number of this line.
/&	inserts the number of bytes examined.

If **/** is followed by any other character, it is interpreted according to SR.

Example using PREFIX and SUFFIX
Example using SINGLE and SUFFIX
Example using AFTER, BEFORE, PREFIX and SUFFIX
SUFFIX ERRORLEVELs

Example using AFTER, BEFORE, PREFIX and SUFFIX:

DIR b:\/a-d/b/s |AFTER : |BEFORE -1. |PREFIX "CALL john c:" |SUFFIX "." >tmp.cmd*

will create a working COMMAND file called TMP.CMD. The COMMAND file will include a line for each file in all directories of B:\ excluding the subdirectories themselves. For the file "B:\DOS\COMMAND.COM," the resulting line would look like this:

```
    CALL john c:\DOS\COMMAND.*
```

Executing TMP.CMD requires that JOHN.CMD exists, too. The code of JOHN.CMD would use %1 to do something with all of the like-named file(s) with any extension.

AFTER
BEFORE
PREFIX
SUFFIX

SUFFIX ERRORLEVELs

Errors are per SR ERRORLEVELS.

UNIQUE

```
Syntax:    UNIQUE [/u]|[[/d][/g][/c][/r][/l][/n][/b]]

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014
```

This filter omits recurrences of lines. Only UNIQUE lines are included. If the input has been SORTed, all recurrences are omitted. If the input has not been SORTed, only contiguous recurrences are omitted, such as multiple contiguous blank lines. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the length of lines is only limited by memory.

Option	Description	Marker	Alias
/U	only include UniQue lines. Omit all duplicate lines.		/Q /1
/D	only include DuPlicate lines. Omit all unique lines.		/M /P
/G	**prefix** each line with the GrAnd ToTAl of recurrences.	-	/T /A
/C	**prefix** each line with its Count of OCCurrenCes.	;	/O
/R	**prefix** each line with its count of RecuRRences.	,	/@
/L	**prefix** each line with its output Line number.	.	/#
/N	**prefix** each line with its Input linE NumbEr. The last recurrence is NumbErEd.	:	/I /E
/B	**prefix** each line with the number of BYtes observed.	=	/& /Y

Slashes, spaces and tabs (**/**) within options are not required.

Numbers generated by the **prefix** options above use a fixed size for the right-justified, space-padded numbers that allows for correct post-sorting. See SR COUNTERS for details.

The Marker character above will be included following each of the numbers generated by **prefix** options above to mark their identity. These **Marker**s are followed by one tab.

If multiple **prefix** options are used, those numbers will be generated in the order shown above.

Example using ECHO2CON, LOWER, UNIQUE and WORDS
Example using AFTER, BEFORE, LOWER, SINGLES and UNIQUE
Example using AFTER, BEFORE and UNIQUE
Example using LOWER, UNIQUE and WORDS
Example 1 using UNIQUE, UPPER and WORDS
Example 2 using UNIQUE, UPPER and WORDS
UNIQUE ERRORLEVELs

Example using UNIQUE, UPPER and WORDS:

UPPER < book.txt | WORDS '- | SORT | UNIQUE /C | SORT /R > words.lst

will create and fill WORDS.LST with a UPPER-case list of all the UNIQUE WORDS contained in BOOK.TXT, including and SORTed according to their frequencies. Furthermore, the most frequent words will be listed first and words with equal frequencies will be listed in Reverse alphabetical order. The list provides a statistical glossary of BOOK.TXT.

UNIQUE ERRORLEVELs:

  8   Illegal combination of options. **/U** option ignored.

Otherwise, errors are per SR ERRORLEVELS.

UPPER

Syntax: UPPER

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter translates every letter to UPPER-case. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the sizes of words are only limited by memory.

Example 1 using UNIQUE, UPPER and WORDS
Example 2 using UNIQUE, UPPER and WORDS
Example using AFTER, BEFORE, UNIQUE, UPPER and WORDS
UPPER ERRORLEVELs

Example using UNIQUE, UPPER and WORDS:

UPPER < book.txt | WORDS ' | SORT | UNIQUE > words.txt

fills WORDS.TXT with an UPPER-case SORTed glossary of all UNIQUE WORDS in BOOK.TXT.

UNIQUE
UPPER
WORDS

UPPER ERRORLEVELs

Errors are per SR ERRORLEVELS.

WORDS

Syntax: WORDS [<embedded>] [<separator>]

(Filter)   (32-bit executable file)

This filter separates WORDS such that there is only one word per line. It does so by converting all sets of white-space and non-alphabetic characters to a new line. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the sizes of words plus white-space and non-alphabetic characters are only limited by memory.

The <EMBEDDED> characters may be included in the words, but words must still start with a letter. Typical text embedded characters are '-. Typical technical embedded characters are _0123456789. Additional characters allowed in DOS 8.3 file names are !~-_0123456789. Additional characters allowed in long file names are .#$%&'()@^`{}.

It also supports an optional SEPARATOR to replace the default new line separator. The replacement may have escape sequences as defined in SR for STRING2.

If this HELP section was filtered by WORDS, the result would start with:

```
WORDS
embedded
separator
Filter
Bit
executable
file
Copyright
c
Gareth
B
Dolby
This
filter
separates
WORDS
such
that
there
is
only
one
word
per
line
```

Example using ECHO2CON, LOWER, UNIQUE and WORDS
Example using LOWER, UNIQUE and WORDS

Example 1 using UNIQUE, UPPER and WORDS
Example 2 using UNIQUE, UPPER and WORDS
Example using AFTER, BEFORE, UNIQUE, UPPER and WORDS
WORDS ERRORLEVELs

Example using AFTER, BEFORE, UNIQUE, UPPER and WORDS:

BEFORE ; <a.asm |AFTER +02 |WORDS _0987654321 |UPPER |SORT |UNIQUE

isolates the operands in assembly file A.ASM. It will segregate the portion that is BEFORE their comments (;) and AFTER 2 tabulations. The WORDS are extracted, allowing for _0987654321 characters and then converted to UPPER-case. They are SORTed and only the UNIQUE ones are shown. The assembler would generate an error message for these operands.

WORDS ERRORLEVELs:

None.
Refer to SR ERRORLEVELS if WORDS reports an error.

SR

Syntax: SR string1 [string2] [<in.file.name] [>out.file.name]

(Filter)   (32-bit executable file)
Copyright (c) Gareth B. Dolby 1997-2014

This filter performs a global Search-and-Replace operation as word processors do, replacing bytes (ASCII characters), but from a command PROMPT on any file type. All portions of the standard input matching STRING1's pattern are replaced by STRING2 in the standard output. If STRING2 is null, all portions matching STRING1's pattern are deleted. All portions not matching STRING1's pattern remain intact, unless the **/K** option is used. BATCH replaceable parameters are allowed. REDIRECTION of the input and output is possible.

The sizes of input and output files are unlimited, while the sizes of FINDINGS are only limited by memory.

Tabs are not automatically expanded. Control Z and NULL are given no special meanings. SR does not change anything except what you specify, so it works on any file type.

The STRINGS on the command line should be enclosed between double quotes ("") if they have any spaces, tabs, >, < or | and while running under Linux, Unix or CYGWIN.

PERSONAE
FINDINGS
ESCAPE SEQUENCES
EXACTIONS

LITERALS
EoL
WILDCARDS
OPTIONS
HOMED MODE
DIAGNOSTIC MODE
FIND MODE
ITEM
MODIFIERS
DRUDGES
DELIMITERS
REPEATERS of DELIMITED FINDINGS
PARSERS
COMPARISONS 27
COUNTERS
JUMP
SPACES AND TABS
UNDEFINED ESCAPE SEQUENCES 30
OPERATION 31
SR ERRORLEVELS 32

PERSONAE

Note that SR has many other personae that simplify its complex command-line parameters. These personae perform preprocessing of SR's command-line parameters. They are named for what they do:

UNIQUE	WORDS	SINGLE	SINGLES	PREFIX
SUFFIX	BEFORE	AFTER	UPPER	LOWER

SR.EXE also has an alias, because there are many programs out there by the same name. You can use SR!.EXE, instead by typing SR! in place of SR. You may delete SR.EXE from your installed directory to allow your other SR.EXE to run normally.

FINDINGS: definition

The FINDINGS grow as a portion of the standard input is found to match STRING1's pattern and are completed when a portion of the standard input is found to match STRING1's entire pattern. That matching portion becomes the FINDINGS.

Example: Positive Replace

Example: Positive Replace

SR "John Kennedy" "Ron Reagan" < letter.txt > PRN:

changes each instance of "John Kennedy" to "Ron Reagan" in LETTER.TXT and prints it. The FINDINGS are *John Kennedy.*

ESCAPE SEQUENCE: definition

If a LITERAL is preceded by a slash (/), then it is no longer a LITERAL, but becomes an ESCAPE SEQUENCE.

ESCAPE SEQUENCES: (STRING1 and STRING2)

STRING1 and STRING2 in the command line may include the ESCAPE SEQUENCES listed in the following sections in UPPER-CASE or lower-case, except UNDEFINED ESCAPE SEQUENCES.

EXACTIONS
WILDCARDS
OPTIONS
MODIFIERS
DRUDGES
UNDEFINED ESCAPE SEQUENCES

EXACTION: definition

An EXACTION is an ESCAPE SEQUENCE, described by 2 or more characters, that represents only one byte.

EXACTIONS

EXACTIONS: (STRING1 and STRING2)

Some ASCII characters are not taken literally, because they edit, delimit, replace or terminate the command line, or they REDIRECT the command's inputs and outputs, so SR includes EXACTIONS to represent these ASCII characters:

EXACTION	Result
/b	Backspace, (BS: /8) does not delete the previous byte.
/t	Tab (/9)
/n	liNe-feed (/10)
/v	| (Vertical tab: /11)
/f	FormFeed (/12)
/r	carriage-Return (/13)
/q	" (Quote: /34)
/:	% (percent: /37)
//	/ (slash: /47)
/[	< (less-than/left angle bracket: /60)
/]	> (greater-than/right angle bracket: /62)
/nnn	a byte in decimal radix; 0 to 255, e.g. /208

LITERALS: definition

LITERALS are WYSIWYG (what-you-see-is-what-you-get) ASCII characters.

EoL: definition

EoLs (End-of-Lines) are the invisible control character(s) that terminate lines in text files. Four types of EoLs are supported. You can use SR and the table of EoL ESCAPE SEQUENCES below to convert from any of 5 EoL types to another and back.

EoL	Operating Systems
"/n"	Unix, GNU/Linux, Multics, OSX, FreeBSD, AIX, Xenix, BeOS, Amiga, RISC.
"/r"	Mac OS through version OS-9, Commodore, Acorn BBC, TRS-80, Apple II.
"/r/n"	Windows, DOS, TOPS-10, RT-11, CP/M, MP/M, TOS, OS/2, Symbian, Palm.
"/n/r"	Acorn BBC and RISC spooled text output.
"/30"	QNX pre-POSIX implementation, before version 4.

The /30 EoL above is not automatically supported by SR, but the other four are automatically recognized while EOL-SENSITIVITY is active. By default, EOL-SENSITIVITY is off, so EoLs are ignored. EOL-SENSITIVITY is activated if STRING1 includes a HOMED MODE OPTION, a LOCATOR MODIFIER or a LINE PARSER.

Example: Macintosh EoL conversion
Example: Positive Delete

Example: Macintosh EoL conversion

```
SR < file.mac > file.unix /R /N
```

converts all carriage-returns to line-feeds, converting Mac text into Unix text, making them more compatible with Windows text.

Example: Positive Delete

```
sr "/n/r" < file.dos > file.txt
```

deletes all blank lines from the Windows-formatted file FILE.DOS or deletes all EoLs from Acorn-formatted files and stores it as FILE.TXT.

WILDCARDS: (STRING1 only)

WILDCARDS are the ESCAPE SEQUENCES that test for a member from a group of possible ASCII characters, or just any byte:

/#	any ASCII digit [0-9] {/g & /!/i & /!/m}
/&	either ASCII binary digit 0 or 1 (48 | 49)
/*	any byte (0-255)
/?	any ASCII digit or letter ignoring case {/# | /i}
/a	any ASCII character (0-127) {/c | /p}
/c	any ASCII Control character (0-31 | 127) {/a & /!/p}

/g	any ASCII Graphic (black-space) character (33-126) {/p & /!/s}
/i	any ASCII letter Ignoring case {/l | /u}
/l	any ASCII Lower case letter [a-z]
/m	any ASCII punctuation Mark {/g & /!/?}
/o	any ASCII Octal digit [0-7]
/p	any ASCII Printable character (32-126) {/a & /!/c}
/s	either ASCII Space or tab (32 | 9)
/u	any ASCII Upper case letter [A-Z]
/w	any ASCII White-space character (9-13 | 32)
/x	any ASCII heXadecimal character ignoring case [0-9 | a-f | A-F]
/$	any quantity of ASCII spaces or tabs (32 | 9)

OPTION: definition

An OPTION is an ESCAPE SEQUENCE embedded anywhere within STRING1 or STRING2, as required, that changes the operating mode of SR for this duration. Any combination of options may be used.

HOMED MODE
DIAGNOSTIC MODE
FIND MODE

HOMED MODE: (STRING1 only)

/h is an OPTION that Homes all tests to align at the beginning of lines. It activates EOL-SENSITIVITY. The test-point will be moved forward to the beginning of the next line automatically, if necessary before testing for STRING1.

Without **/H**, SR tests relentlessly. For the command SR "/e/-/@9" "/o/d/r/n", that relentless testing finds recurrences where the next line(s) match the remainder of the first line. **/H** solves this in the command SR "/H/e/-/@9" "/o/d/r/n".

DIAGNOSTIC MODE: (STRING1 only)

/y is an OPTION saying "Yes, do display all counter values and the ERRORLEVEL on the console after execution." Below is an example. Your results will vary.

```
SR (Copyright Gareth B. Dolby 10 2013)
received 80666 bytes,
transmitted 99076 bytes,
counted 2630 lines,
replaced 2630 findings,
used 128 KB of RAM and returned
errorlevel 0.
```

Otherwise, SR remains quiet unless a fatal error occurs.

The ERRORLEVEL will reveal warnings that might otherwise not be displayed. Decode the meanings of warnings and errors at the end of this document under ERRORLEVELS.

Example: DIAGNOSTIC MODE

Example: DIAGNOSTIC MODE

SR /y/h/=/-60//r/n <book1.doc >PRN:*

deletes lines with DOS EoLs and exactly 60 other characters in BOOK1.DOC and prints the remainder. It also displays counter values and the ERRORLEVEL on the console (**/Y**) to help debug the search-and-replace process.

FIND MODE: (STRING2 only)

/k is an OPTION that Kills all standard output that is not described by STRING2. Otherwise, SR transmits bytes as-is that fail the tests of STRING1.

ITEM: definition (STRING1 only)

ITEMS are LITERALS, EXACTIONS, WILDCARDS, DELIMITERS, PARSERS and COMPARISONS. An ITEM can be expressed with one character, as in “Z”, or MODIFIED many times, as in "/=/!/+5/-10/\Z".

MODIFIERS: definition (STRING1 only)

MODIFIERS modify the behavior of ITEMS. They may be combined in any order to modify one ITEM as long as all MODIFIERS precede their ITEM. A slash is required for each MODIFIER, EXACTION, WILDCARD, DELIMITER, REPEATER, PARSER and COMPARISON.

OPTIONS cannot be MODIFIED, but since OPTIONS are not ITEMS, MODIFIERS and OPTIONS ignore each other.

MODIFIERS at the end of STRING1 are ignored.

ALL MODIFIERS

ALL MODIFIERS: (STRING1 only)

/= /_	LOCATES this item, instead of testing for it.
/+[n]	MIN: continues for N (decimal) occurrences or more of this ITEM.
/-[n]	MAX: continues for N (decimal) occurrences or less of this ITEM.
/!	tests for NOT this ITEM.
/	tests BACKWARD for this ITEM or mirrors this PARSER.

LOCATOR MODIFIERS
MIN and MAX MODIFIERS
NOT MODIFIER
BACKWARD MODIFIER

LOCATOR MODIFIERS:

LOCATE MODIFIERS LOCATE the MIN-MAXth occurrence of their ITEM on the remainder of this line and position the next test-point while keeping all bytes prior to their ITEM as a part of the FINDINGS.

LOCATORS preserve those bytes before their ITEM to be repeated, tested, deleted, etc. Otherwise, SR ignores EoLs and transmits bytes as-is that fail the tests of STRING1. LOCATORS break this behavior.

The two LOCATORS work the same as each other, except for where they position the next test-point. **/_** stops before its ITEM in the direction used while **/=** stops after its ITEM in the direction used (forward or BACKWARD).

/_	LOCATES the MIN-MAXth ITEM on this line and stops before it.
/=	LOCATES the MIN-MAXth ITEM on this line and stops after it.
/_/=	/= is ignored if /_ is also used to MODIFY the same ITEM.

LOCATORS fail if the end of this line occurs first or its ITEM is **/R** or **/N**, due to having EOL-SENSITIVITY.

MIN and MAX MODIFIERS: (STRING1 only)

Without MIN or MAX, only 1 ITEM is tested. MIN and MAX modify the required quantity. MAX sets the maximum required quantity. MIN sets the minimum required quantity. They will test for or locate an ITEM more than once and allow for a quantity of 0.

/+[n] is the MIN entry.
/-[n] is the -MAX entry. It is used as a positive number.

The range of MIN is 0 to -MAX. A MIN of 0 prevents failure.

The range of -MAX is MIN to the size of available memory (effectively infinity).

MIN and MAX Dynamic Defaults

MIN and MAX Dynamic Defaults:

The MIN and MAX MODIFIERS' values have dynamic defaults according to the following rules:

1) MIN and MAX default to 1 and are then subject to rules 2 through 7 below, so "Z" tests for exactly 1 "Z" byte.

2) If both MIN and MAX values are specified, then nothing defaults, so "/+3/-7Z" tests for 3 to 7 occurrences of "Z" bytes.

3) If a value for MAX is specified, but MIN is not invoked, then MIN defaults to MAX, so "/-7Z" tests for exactly 7 occurrences of "Z".

4) If MAX is used without a value, then MAX defaults to the amount of memory available, so "/-Z" tests for 1 or more occurrences of "Z".

5) If a value for MIN is specified, but MAX is not invoked, then MAX defaults to the amount of memory available, so "/+3Z" tests for 3 or more occurrences of "Z".

6) If MIN is used without a value, then MIN defaults to 0, so "/+Z" tests for 0 to 1 occurrences of "Z" and cannot fail.

7) MIN may not be greater than -MAX. If violated, the last value specified will be used for both MIN and MAX.

These 7 rules cover 9 situations for the MIN and MAX values, summarized in the matrix below to search for "Z". They provide 8 different ways to MODIFY an ITEM's search quantity:

STRING1	**MAX**		
MIN	(none)	**/-**	**/-7**
Z	1 "Z"	1 to infinity "Z"	7 "Z"
/+Z	0 or 1 "Z"	0 to infinity "Z"	0 to 7 "Z"
/+3Z	3 to infinity "Z"	3 to infinity "Z"	3 to 7 "Z"

Infinity represents the amount of memory available.

/-7/+3Z will pass FINDINGS of ZZZ, ZZZZ, ZZZZZ, ZZZZZZ or ZZZZZZZ.

Note: Large MIN and MAX values influence the amount of memory SR uses.

NOT MODIFIER:

/! is the NOT MODIFIER, which inverts the test result to test for anything other than this ITEM. It also makes LITERALS and EXACTIONS act like WILDCARDS. "/!/+2/-8Z" tests for 2 to 8 occurrences of bytes that are NOT "Z".

Example: Negative Replace

Example: Negative Replace

SR /+1/!/i /r/n < letter.txt | sort | UNIQUE displays a SORTED list of the UNIQUE words found in LETTER.TXT, but cannot recognize words containing punctuation.

BACKWARD MODIFIER:

/\   is the BACKWARD MODIFIER, which tests BACKWARD (left), instead of forward (right). BACKWARD tests are limited to STRING1's FINDINGS so far, so do not test BACKWARD until you have first tested forward. Attempts to test BACKWARD behind the first ITEM of STRING1 will stop.

When combined with MIN and MAX, a BACKWARD test could reach the first ITEM, again, as in "/\\/+0/*", but use "/j" as a short-cut to get to that first ITEM, instead.

DRUDGES: definition

DRUDGES are ITEMS that do tedious, menial, or unpleasant work. Most DRUDGES process many bytes, instead of one, so DRUDGES are much faster.

DELIMITERS
REPEATERS
PARSERS
COMPARISONS
COUNTERS
JUMP
SPACES AND TABS
STRING2 DRUDGES

DELIMITERS: (STRING1 only)

/d   is a DRUDGE that enumerates locations in the FINDINGS with up to 10 Delimiting markers from 1 to 10. If exceeded, the last DELIMITER in STRING1 will relocate the tenth DELIMITER to its location.

The start and end of the FINDINGS are already DELIMITED.

BACKWARD MODIFIED DELIMITERS (**/\\/D**) will relocate the last DELIMITER to this new location. If there were no prior DELIMITERS, one will be created.

MIN, MAX, NOT and LOCATE MODIFIERS of DELIMITERS are ignored.

REPEATERS of DELIMITED FINDINGS: (STRING2 only)

REPEATERS of DELIMITED FINDINGS: (STRING2 only)

VERBATIM REPEATERS
PROCESSED REPEATERS

VERBATIM REPEATERS: (STRING2 only)

/d[#] is a DRUDGE that REPEATS the DELIMITED FINDINGS from the original standard input to the standard output. # = 0 to 9. Default # = 0.

The repetition stops at the next DELIMITER:

Use	to REPEAT the FINDINGS found...
/d0	before DELIMITER 1 of STRING1
/d1	between DELIMITERS 1 and 2 of STRING1
/d2	between DELIMITERS 2 and 3 of STRING1
/d3	between DELIMITERS 3 and 4 of STRING1
/d4	between DELIMITERS 4 and 5 of STRING1
/d5	between DELIMITERS 5 and 6 of STRING1
/d6	between DELIMITERS 6 and 7 of STRING1
/d7	between DELIMITERS 7 and 8 of STRING1
/d8	between DELIMITERS 8 and 9 of STRING1
/d9	between DELIMITERS 9 and last of STRING1
/d	before DELIMITER 1 of STRING1

If STRING1 has fewer than # DELIMITERS (or none), then "/d[#]" will REPEAT the entire FINDINGS.

If STRING1 has exactly # DELIMITERS, there is no user-defined terminus DELIMITER, so repetition will stop at the end of the FINDINGS, e.g. if STRING1 has exactly 5 DELIMITERS, then "/d5" will REPEAT the remainder of the FINDINGS found after that fifth DELIMITER.

If STRING1 has ten or more DELIMITERS, then that portion of the standard input found after the tenth DELIMITER cannot be REPEATED alone. The tenth DELIMITER can only serve as the terminus for "/d9" and "/@9".

If STRING1 has more than ten DELIMITERS, then the tenth DELIMITER will mark the last of them, such that "/d9" and "/@9" will use that portion of the standard input found between DELIMITER 9 and the last DELIMITER in STRING1.

Example: Positive Replace and Repeat
Example: LOCATING bytes
Example: Negative Repeat with Delete
Example: Negative Repeat and Insert using BACKWARD

Example: Positive Replace and Repeat

```
SR John/d/+1/w/dKennedy Ron/d1Reagan < letter.txt
```

changes each instance of "John Kennedy" to "Ron Reagan" while preserving the white-space. "Kennedy" could be indented on the next line after "John".

Example: LOCATING bytes

```
dir |SR "/h/=/-41/*/\/-31/*" "/d9/t" |sort
```

inserts a tab after the tenth byte on each line from the DIR command that has at least 41 characters and sorts it before it is displayed on the terminal. Listings from DIR and LS commands can be re-formatted many ways.

Example: Negative Repeat with Delete

```
SR /h/=/-9/*/d/!/g /d0 < letter.txt
```

deletes the tenth byte from each line in LETTER.TXT unless it is graphic (/!/g) and displays it on the terminal. Shorter lines are not changed.

Example: Negative Repeat and Insert using BACKWARD

```
SR "/!/-81/r/\/+0/g" "/d9/r/n" <letter.txt
```

inserts a DOS EoL (/r/n) before the word which occupied the 81st byte wherever LETTER.TXT continued for 81 bytes without any EoLs and displays it on the terminal. Words will not be split. This is the operation of word wrapping to prevent long lines from chopping words onto 2 lines.

PROCESSED REPEATERS: (STRING2 only)

These 3 DRUDGES change the case of letters within the DELIMITED FINDINGS and then REPEAT those DELIMITED FINDINGS using the same rules as /d[#]. Non-alphabetic bytes are not changed.

/c[#]	changes the Case of all letters.
/l[#]	changes upper-case letters to Lower-case.
/u[#]	changes lower-case letters to Upper-case.

The case is changed in the buffer, so accessing them again will find the case changes.

Example: Positive Repeat and Insert While Changing Case

Example: Positive Repeat and Insert While Changing Case

```
DIR /s/b | SR "/i/d:\/d/e" "/L2 on /u0/d3"
```

displays file names first in lower-case and disks last in upper-case from the "DIR /S/B" command onto the terminal. "C:\CONFIG.SYS" becomes "config.sys on C" and "D:\DOS\XCOPY.EXE" becomes "dos\xcopy.exe on D", etc. Listings from DIR and LS commands can be re-formatted many ways.

PARSERS: definition (STRING1 only)

PARSERS LOCATE a specific pattern, place a DELIMITER there, position the test-point beyond that pattern and keep all bytes prior to it as part of the FINDINGS.

PARSERS continue to perform their function, even after the supply of 10 DELIMITERS has been placed. The tenth DELIMITER will point to the last DELIMITED byte, even if it was the 50th DELIMITER.

Parsing Lines
Parsing Words

Parsing Lines:

/e is a DRUDGE that parses the remainder of this line with EOL-SENSITIVITY.

Forward tests parse the End of this line, identified by the next EoL. A DELIMITER is placed at the End of this line, before its EoL. The next test will be at the beginning of the next line.

BACKWARD tests parse the beginning of this line, identified by the previous EoL. A DELIMITER is placed at the beginning of this line, after the previous line's EoL. The next test will be at the end of the previous line.

The end of the file is not an EoL.

LOCATE MODIFIERS of line parsers are ignored.

Uses: Comparing Entire Lines
Example: Positive Delete using a Homed Line Parser
Example: Adding Prefixes and Suffixes
Example: Adding Odd/Even Prefixes and Suffixes
Example: Positive Insert a Suffix
Example: Positive Delete Blank Lines
Example: Line Swapping

Uses: Comparing Entire Lines

```
SR "/h/e/@0"
or
SR "/h/e/@"
```

will compare one entire line against the next, excluding their EoL byte(s), while

```
SR "/h/e/@9"
```

will include their EoL byte(s), verifying that the lines end identically.

```
SR "/h/e/+1/@9"
```

will compare one entire line against the next line... and the next line, stopping when one or its EOL differs. Recurrences are counted.

Example: Positive Delete using a Homed Line Parser

```
sr < input.c "/h/////e" > file.c
```

deletes all lines that begin with // from INPUT.C and stores the result in FILE.C. This deletes one type of comment-only lines from C, C#, C++, Java, etc. source files.

Example: Adding Prefixes and Suffixes

```
SR <document1.txt >>document2.txt "/e" "PREFIX/d0SUFFIX/d1"
```

adds "PREFIX" to the beginning and "SUFFIX" to the end of all lines found in DOCUMENT1.TXT and appends this to the end of DOCUMENT2.TXT.

Example: Adding Odd/Even Prefixes and Suffixes

```
SR <document1.txt >>document2.txt "/-2/e" "PREFIX/d0/d1SUFFIX/d2"
```

adds "PREFIX" to the beginning of odd-numbered lines and "SUFFIX" to the end of even-numbered lines found in DOCUMENT1.TXT and appends this to the end of DOCUMENT2.TXT.

Example: Positive Insert a Suffix

```
dir /b | SR /e "CALL process /d9" > tmp.bat
```

creates TMP.BAT, a working BATCH file which includes a line for each file found by the DIR /B COMMAND. A possible line might be:

```
CALL process AUTOEXEC.BAT
```

Executing TMP.BAT requires that PROCESS.BAT exists, too. The code of PROCESS.BAT would use %1 to do something with the named file(s).

Example: Positive Delete Blank Lines

```
SR   "/E/D/-/@1" "/D/D1"   <any.txt   >book.txt
```

deletes blank lines of any supported EoL type. SR compares EoLs in ANY.TXT to the next character(s) and omits all recurrences from its output: BOOK.TXT.

Example: Line Swapping

```
SR "/-5/e" <document1.txt "/d3/d4/d0/d1/d2/d5"
```

swaps lines 4 and 5 with lines 1, 2 and 3 in DOCUMENT1.TXT, repeating every 5 lines. If DOCUMENT1.TXT has 9 lines, only one such swap will occur.

Parsing Words

/{[set]} is a DRUDGE that parses words.

Words must begin with a letter and may contain any number of letters and the optional [SET] of ASCII characters enclosed within {}. Words end at the first character that is not an ASCII letter or member of [SET].

Forward tests parse the beginning of the next word, identified by the next letter, so they always begin with a letter. A DELIMITER (/d) is placed at the beginning of the word, separating it from whatever preceded it. The next test will be after the end of the parsed word.

BACKWARD tests parse the end of the previous word, identified by the previous letter, so they always end with a letter. A DELIMITER (/d) is placed at the end of the word, separating it from whatever followed it. The next test will be at the beginning of the parsed word.

The [SET] may not contain ESCAPE SEQUENCES; only LITERALS. All [SETs] in all word parsers will be combined.

Uses: Parsing Words
Example: Finding Sentences

Uses: Parsing Words

```
      SR "/{}" "/d1/r/n"
```

parses simple words onto each line.

```
      SR "/{'-}" "/d1/r/n"
```

parses common words and phrases.

```
      SR "/{_0123456789}" "/d1/r/n"
```

parses labels used in programming languages.

```
      SR "/{_()[]0123456789}" "/d1/r/n"
```

includes arrays and functions used in programming languages.

Example: Finding Sentences

```
SR "/{+-`'&/@#$0123456789%() ,;:}" "/k/d1./r/n" <any.type |SR /n/+1/-
2/!../r
```

displays whole sentences from the same line in FILE ANY.TYPE, containing common words and phrases, omits everything else, omits tiny sentences, adds periods and DOS EoLs. EOL-SENSITIVITY is off, so if ANY.TYPE is a binary file, all its embedded sentences are displayed, while binary junk is excluded.

Note that the slash within braces is taken literally.

COMPARISONS: (STRING1 only)

/@[#] is a DRUDGE, that compares the next standard input against the DELIMITED FINDINGS already found @ DELIMITER #, expecting a complete and perfect recurrence. # = 0 to 9. Default # = 0. You cannot make COMPARISONS until after you have developed some FINDINGS to compare against.

Successful COMPARISONS stop at the next DELIMITER (DELIMITER #+1). If there are no more DELIMITERS, successful COMPARISONS stop at the end of the original FINDINGS. In both cases, the FINDINGS then grow.

If there has been fewer than # DELIMITERS so far, the COMPARISON will compare against the entire original FINDINGS. In fact, COMPARISONS MODIFIED by MIN or MAX often need to compare the entire FINDINGS or begin at the last DELIMITER.

The MIN and MAX MODIFIER values of COMPARISONS refer to the number of recurrences, not occurrences nor bytes.

The next test after a successful COMPARISON will be beyond the last recurrence.

The COMPARISON fails if any of the compared standard input differs from their respective DELIMITED byte, DELIMITER # refers to the same byte or DELIMITER # is at or beyond the terminus DELIMITER. The latter of these failures could occur if you built duplicate or BACKWARD DELIMITERS.

For NOT MODIFIED COMPARISONS; the COMPARISON succeeds and the test fails if all of the standard input matches their respective DELIMITED bytes. If any byte does not match, the test passes and the next test will be at the first compared byte, even if that byte matched.

BACKWARD and LOCATE MODIFIERS of COMPARISONS are ignored.

Uses: Positive Compare Adjacent Letters
Uses: Negative Compare
Example: Positive Delete Partially-Identical Lines
Example: Negative Delete Partially-Differing Lines
Uses: Repeating Compares

Uses: Positive Compare Adjacent Letters

```
SR "/u/@0"
or
SR "/u/@"
```

will find any upper-case letter (/U) and then compare the next byte to it.

Uses: Negative Compare

```
      SR "AB/d/#/dEF/!/@1"
```

will compare NOT (/!/@1) against the digit found by /# after the first DELIMITER (/D), failing if it is the same. A FINDING might be "AB6EF7".

Example: Positive Delete Partially-Identical Lines

SR "/h/=,/d/e/d/=,/@1/@2/j3" <any.csv |more

deletes entire lines in ANY.CSV where the second line matches the first line beyond their first commas. Lines with no commas are ignored. The remaining lines are displayed one page at a time. This deletes duplicate records with different data in the first field, since the second line is preserved.

Example: Negative Delete Partially-Differing Lines

sr "/h/=,/d/e/=,/!/@1/e" <letter2.csv >letter3.csv

deletes lines in LETTER2.CSV if the two lines differ after their first commas and stores the remainder in LETTER3.CSV, including lines with no commas. This deletes pairs of duplicate records with different data in the second field.

Uses: Repeating Compares

```
      SR "AB/dCD/d/i/dGH/-5/@2"
```

will compare against the letter found by /I after the second DELIMITER (/d), repeatedly, failing upon a mismatch or stopping after the fifth recurrence. It would find "ABCDeGHeeeee". Recurrences are counted. This example would count 5 recurrences.

COUNTERS:(STRING2 only)

Several STRING2 DRUDGES inject numerical values into the standard output. These values are derived from COUNTERS built into like DRUDGES in STRING1, or are always active, as shown below:

STRING1	STRING2	Minimum	COUNTER
FINDINGS	**/#**	1	Replacements
/{	**/{ | /w**	1	Words
/@	**/@ | /m**	1	Recurrences
/@	**/o**	2	Occurrences (Recurrences+1)
/@	**/g**	1	Grand Total Recurrences
	/e	0	Lines
	/&	1	Bytes

/#   is the ***Replacements*** counter, which always counts FINDINGS that have been Replaced so far.

/{ or **/w**   is the ***Words*** counter, which counts all Words parsed so far by all WORD PARSER DRUDGES (/{set} in STRING1).

/@ or **/m**   is the ***Recurrences*** counter, which counts REPLACED recurrences found in these FINDINGS by all COMPARISON DRUDGES (/@ in STRING1). The Recurrences counter resets to zero after injection and test failures, before counting the next FINDINGS. Therefore, it does not count successful COMPARISONS where another ITEM in STRING1 failed. Furthermore, recurrences are only counted if the ***Replacements*** counter is also incremented.

/o   is the ***Occurrences*** (Recurrences+1) counter, which injects one more than the ***Recurrences*** counter does.

/g   is the ***Grand Total Recurrences*** counter, which counts REPLACED Recurrences in all FINDINGS observed so far by all COMPARISON DRUDGES (/@ in STRING1). The ***Grand Total Recurrences*** is the sum of all ***Recurrences*** counts so far.

/e   is the ***Lines*** counter, which always counts the EoLs observed so far up to the end of the current FINDINGS. It never counts the same line twice. It does not count all EoLs in binary or mixed EoL files.

/&   is the ***Bytes*** counter, which always counts the bytes observed so far up to the end of the current FINDINGS. It provides the offset to the next byte. Its value wraps back to zero upon ignored overflows.

The **Minimum** column in the table above estimates the lowest injectable value for a normal, warning-free execution. It is this way because all COUNTERS are incremented before injection, so they include the current FINDINGS.

All COUNTER values begin from zero and are written to the standard output using the same number of ASCII characters with the sortable, decimal unsigned integer right-justified and space-padded. Include custom separators to add clarity.

JUMP: *(STRING1 only)*

/j[#] is a DRUDGE that Jumps to DELIMITER #.   # = 0 to 9. Default # = 0.

/j[#] is used to redirect the next test or remove previously-matched bytes from the FINDINGS.

If there have been fewer than # DELIMITERS so far, /J# will Jump to the beginning of the FINDINGS, as do /J0 and /J.

All MODIFIERS of JUMPS are ignored.

SPACES AND TABS: *(STRING1 only)*

Spaces and tabs can look the same. This is solved by the aforementioned /S WILDCARD. They are also used in groups to indent, but you can't see how many nor in what order they are. This is solved using "/+1/S", which is the same as "/-/S". Spaces and tabs are also used as delimiters to separate and align columns of data for easy reading, but difficult parsing. This is solved using the /$ WILDCARD DRUDGE.

/$ is a WILDCARD DRUDGE ITEM that tests for groups of one or more $paces or tabs (32 or 9), just like /+1/S would. The difference is that /$ ignores MIN and MAX MODIFIERS to reserve them for use with a LOCATE MODIFIER. This gives /$ the power to LOCATE the nth group of one or more $paces or tabs on this line.

The next test will begin at the first byte that is not $pace nor tab.

NOT MODIFIED **/$** tests will only advance one byte.

Example: Scanning Across Multiple Groups of White Space

Example: Scanning Across Multiple Groups of White Space

SR "/h/=/-3/$/d/e" <table1.tsv "/d1/d2" >>table2.tsv

will delete all text from the beginning of each line in TABLE1.TSV up to and including the first 3 groups of $paces and tabs and append the remainder to TABLE2.TSV. This can be used to remove the first 3 tabulated records from a database.

UNDEFINED ESCAPE SEQUENCES:

Slashes are ignored if the next character is an UNDEFINED ESCAPE SEQUENCE, as listed in 3 groups below. The character after the slash will be taken literally.

UNDEFINED ESCAPE SEQUENCES: (STRING1 and STRING2)
UNDEFINED ESCAPE SEQUENCES: (STRING1 only)
UNDEFINED ESCAPE SEQUENCES: (STRING2 only)

UNDEFINED ESCAPE SEQUENCES: (STRING1 and STRING2)

/~	invert.
/^phrase^	parses this line at its Nth occurrence of PHRASE.
/.	parses the next sentence, identified by a period (.).
/()	parses the next phrase, enclosed inside parentheses (phrase).
/,	parses the next comma-separated-value.
/'	
/%	gets confused with BATCH replaceable parameters.
/<	gets confused with redirected standard input.
/>	gets confused with redirected standard output.

/|	gets confused with piped standard input and output.
/"	gets confused with delimiters.
/	gets confused with delimiters.
/;	
/	ending STRING1 or STRING2 with just one / is a syntax error.
/n	OBSOLETE: Carriage-Return or Line-Feed, now just Line-Feed.
/z	OBSOLETE: reset COUNTERS to Zero, now automated.

UNDEFINED ESCAPE SEQUENCES: (STRING1 only)

/e	OBSOLETE: imaginary End location, now End-of-line (EoL) parser.
/h	OBSOLETE: imaginary Home location, now Homes tests.
/k	Keep whole lines in input buffer.

UNDEFINED ESCAPE SEQUENCES: (STRING2 only)

/a	/h	/i	/j		
/p	/s	/x	/y		
/!	/+	/-	/\	/?	/*

OPERATION:

AND/OR LOGIC of TESTS: (STRING1 only)
GOING BACK
STRING2 DRUDGES

AND/OR LOGIC of TESTS: (STRING1 only)

SR quickly and relentlessly tests for the first ITEM in STRING1. Each additional ITEM implies a logical AND related to all other ITEMS. Each WILDCARD implies a logical OR for its ITEM. Use SR once for each logical OR that cannot be handled by WILDCARDS or NOT.

GOING BACK

Many repercussions arise from using JUMPS and BACKWARD MODIFIERS. These can cause the FINDINGS to exclude previously-observed bytes. These backed-over bytes will not be REPLACED. Instead, they will be included in the next test.

Going back also allows DELIMITERS and PARSERS to build BACKWARD DELIMITERS. Attempts to REPEAT portions between BACKWARD DELIMITERS yields nothing. Attempts to COMPARE portions between BACKWARD DELIMITERS fails.

A DELIMITER beyond the FINDINGS allows you to REPEAT a portion outside the FINDINGS and test that portion again to REPEAT it twice.

Think of the test-point as stopping between bytes when using DELIMITERS, PARSERS, JUMPS and BACKWARD tests.

Example: JUMP back, REPEAT and observe again

sr "/y/h/=,/d/e/d/_,/d,/@1/@2/j3" "/d0{/d3}/D1/D2" <A.csv >B.csv

searches for two lines that match perfectly after their first commas (/@1/@2). It then copies the text from the second line before its first comma into the first line, after its first comma. The copy is enclosed in braces {/d3}.

All counters and the ERRORLEVEL are displayed on the terminal (/Y).

The JUMP (/j3) puts the test-point back to the beginning of the second line so the FINDINGS encompass only the first line. So, the next test will compare the second and third lines. Without the JUMP, the next test would compare the third and fourth lines. This also means that the copy operation of /d3 was legally taken from outside the final FINDINGS.

This can be used to merge similar records with differing first fields.

STRING2 DRUDGES

All DRUDGES in STRING2 (REPEATERS and COUNTERS) share a common resource, which limits their total quantity to 10.

FIND & SORT

FIND and SORT are two filters that are built into most operating systems. They are used in many examples herein to demonstrate the use of other filters. But, they do not work the same in all environments!

SR ERRORLEVELS:

ERRORLEVELS below 256 are added together and are non-fatal warnings. ERRORLEVELS above 256 are fatal, but some operating systems truncate ERRORLEVELS to a byte.

0	STRING1 successfully replaced by STRING2.
1	STRING1 not found. Nothing changed.
2	Too many DELIMITERS. DELIMITER 10 = last DELIMITED byte.
4	No standard output transmitted. Everything deleted.
8	Slash ignored.
16	Too many DRUDGES in STRING2.
32	MAX less than MIN in /-n and /+n pair. Used latter.
64	Expected }.
128	Expected a number.
257	Operation not permitted
258	No such file or directory

259	No such process
260	Interrupted system call
261	Input/Output error
262	No such device or address
263	Argument list too long
264	Executable file format error
265	Bad file descriptor
266	No child processes
267	Resource temporarily unavailable
268	Cannot allocate memory / Not enough space
269	Permission denied
270	Bad address
271	Block device required / Unknown error
272	Device or resource busy
273	File exists
274	Invalid cross-device link / Improper link
275	No such device
276	Not a directory
277	Is a directory
278	Invalid argument
279	Too many open files in system
280	Too many open files
281	Inappropriate ioctl for device
282	Text file busy / Unknown error
283	File too large
284	No space left on device
285	Illegal seek
286	Read-only file system
287	Too many links
288	Broken pipe
289	Numerical argument out of domain
290	Number out of range
291	No message of desired type / Unknown error
292	Identifier removed / Resource deadlock avoided
293	Channel number out of range / Unknown error
294	Level 2 not synchronized / File-name too long
295	Level 3 halted / No locks available
296	Level 3 reset / Function not implemented
297	Link number out of range / Directory not empty
298	Protocol driver not attached / Illegal byte sequence!
299	No CSI structure available
300	Level 2 halted
301	Resource deadlock avoided
302	No locks available
303	*Argument list too short
304	*Infinite loop aborted
305	*Findings length exceeded available memory
306	Invalid exchange
307	Invalid request descriptor
308	Exchange full
309	No anode
310	Invalid request code

311	Invalid slot
312	File locking deadlock error
313	Bad font file format
316	Device not a stream
317	No data available
318	Timer expired
319	Out of streams resources
320	Machine is not on the network
321	Package not installed
322	Object is remote
323	Link has been severed
324	Advertise error
325	Srmount error
326	Communication error on send
327	Protocol error
330	Multihop attempted
331	Inode is remote (not really error)
332	RFS specific error
333	Bad message
335	Inappropriate file type or format
336	Name not unique on network
337	File descriptor in bad state
338	Remote address changed
339	Cannot access a needed shared library
340	Accessing a corrupted shared library
341	.lib section in a.out corrupted
342	Attempting to link in too many shared libraries
343	Cannot exec a shared library directly
344	Function not implemented
345	No more files
346	Directory not empty
347	File name too long
348	Too many levels of symbolic links
351	Operation not supported
352	Protocol family not supported
360	Connection reset by peer
361	No buffer space available
362	Address family not supported by protocol
363	Protocol wrong type for socket
364	Socket operation on non-socket
365	Protocol not available
366	Cannot send after transport endpoint shut-down
367	Connection refused
368	Address already in use
369	Software caused connection abort
370	Network is unreachable
371	Network is down
372	Connection timed out
373	Host is down
374	No route to host
375	Operation now in progress
376	Operation already in progress

377	Destination address required
378	Message too long
379	Protocol not supported
380	Socket type not supported
381	Cannot assign requested address
382	Network dropped connection on reset
383	Transport endpoint is already connected
384	Transport endpoint is not connected
385	Too many references: cannot splice
386	Too many processes
387	Too many users
388	Disk quota exceeded
389	Stale NFS file handle
390	Not supported
391	No medium found
392	No such host or network path
393	File-name exists with different case
394	Invalid or incomplete multi-byte or wide character
395	Value too large for defined data type
396	Operation cancelled
397	State not recoverable
398	Previous owner died
399	Streams pipe error