

SPFLite - V10.1.8351
2018-12-16

Table of contents

Welcome to SPFLite	9
Support and Contacts	13
Major Features of SPFLite	14
Installing SPFLite	21
Creating a Portable Version	23
Differences between SPFLite and IBM ISPF	25
Use of the Mouse	29
Default Key Definitions	31
Customizing SPFLite	35
Options - General	36
Options - File Manager	43
Options - Submit	47
Options - Screen	50
Options - Keyboard	57
Options - Status	59
Options - Schemes	60
Options - Hi-Lites	61
Keyboard Customization and Keyboard Macros	64
Key Mapping Overview	67
Using the KEYMAP Dialog	72
Keyboard Macros	82
Working with SPFLite	88
Starting and Ending SPFLite	89
Status Bar Contents	94
Basic Edit Functions	98
Edit Boundaries	101
Scrolling the Text	102
Finding and Changing Data	104
Specifying a Picture or Format String	117
Specifying a Regular Expression	125
Working with Mapping Strings	137
Mapping String Comprehensive Guide	138
Preface	139
Fundamental Concepts	140
Basic features of data mapping	142
Understanding how string mapping works	144
Understanding dot notation	146
How a mapping string is specified	148
Guidelines for writing mapping strings	149
Overview of supported mapping items	151
Mapping String Command Topics	159
End-relative column references	160
Mixed-mode end-relative column reference item	162
External mapping string item – SV	163
Dynamic mapping string item – DV	165
Specifying an alignment item – L / R / C	167
Trimming items – T / TL / TR	170

Padding items – P / PL / PR	171
Character reversal item – RV	172
Zero suppression item – Z	173
Character-replacement item – RC	175
String replacement items – RA / RL / RR	178
Character conversion items – AX / XA / EX / XE	181
Numeric conversion items – DD / DX / XD / XX	183
Sequence items – SD / SX	187
Exchange Items - X	189
Character-move item – M	193
Delete character-columns item – D	197
Delete character-values item – DC	198
Mapping String Calculation Operands	200
Mapping String Quick Reference Guide	217
General information	218
Character enumeration guide	220
Auto-Reference and Auto-Copy features	224
Inserted text	225
Text Case Handling	226
Command-code syntax	230
General examples	232
Examples of DX and XD numeric conversion commands	234
Command Specific Syntax and Examples	236
Working with Sequence Numbering	256
Using the SPFTest Program	260
Shifting Data	265
Defining Tabs and Column Markers	271
Windows Clipboard, Cut and Paste	274
Saving the Edit STATE	278
Working with Tab Pages	281
Working with Command Chaining	286
Working with the File Manager	292
Working with File Lists	307
Working with File Profiles	312
Working with Excluded Lines	318
Working with User lines	334
Working with Multi-Edit Sessions	343
Working with Line Labels	351
Working with Line Tags	357
Working with NOTE and xNOTE Lines	370
Working with Word and Delimiter Characters	376
Working with Power Typing Mode	379
Working with Enumerations	393
Using AUTOFAV to add to File Lists	398
Working with the SUBMIT Command	400
Writing a MACRO for a macro controlled string CHANGE	406
Print Screen Functions	409
Creating and Replacing Data Files	411
Including (Copying) Another File	412
Word Processing Support	413

Working with Virtual Highlighting Pens	421
Working with SPLIT and JOIN Commands	425
Performing Searches with Find in Files	439
Command Keys and Substitution Strings	442
Handling Non-Windows Text Files	444
Working with Unicode Mapping Files	451
Managing Line Lengths	456
Working with Read Only Files	458
Handling External File Changes	462
Working with the LINE Primary Command	464
Line Commands	473
Rules for Entering Line Commands	475
Extended Line Command Modifiers	477
A - After Destination	480
AA - After Block	481
B - Before Destination	483
BB - Before Block	484
BNDS - Display Bounds Line	486
C / CC - Copy Lines	489
COLS - Display Columns Line	490
D / DD - Delete Lines	491
F - Display First Lines in Excluded Range	492
G / GG - Glue Lines Together	493
H / HH - HERE Destination	496
I - Insert New (Temporary) Blank Lines	497
J / JJ - Join Lines Together	498
L - Display Last Lines in Excluded Range	500
LC / LCC - Lower-Case Lines	501
M / MM - Move Lines	502
MARK - Set Column Markers	504
MASK - Set the Insert line model	506
MD / MDD - Make Data Line	508
MN / MNN - Make Note Line	510
N - Insert New (Permanent) Blank Lines	512
NOTE - Insert NOTE or xNOTE Lines	513
O / OO - Overlay Lines	515
OR / ORR - Overlay-Replace Lines	521
PL / PLL - Pad Lines to Length	523
R / RR - Repeat Lines	524
S / SS - Show lines	525
SC / SCC - Sentence-Case Lines	526
SI - Show Indentation	527
T / TT - Select Text Lines	528
TABS - Display TABS Line	531
TB / TBB - Text-Break Lines	532
TC / TCC - Title-Case Lines	534
TF / TFF - Text-Flow a Paragraph	535
TG / TGG - Text Glue Lines	537
TJ / TJJ - Text Join Lines	539
TL / TLL - Truncate Lines to a Length	541

TM / TMM - Text Margin	542
TR / TRR - Trim Trailing Blanks	545
TS - Text-Split a line	546
TU / TUU - Toggle User status of lines	548
TX / TX - Toggle Excluded status of lines	549
U / UU - Mark User lines	550
UC / UCC - Upper-Case Lines	553
V / VV - Revert User Line status	554
W / WW - Swap Lines	557
WORD - Display Valid Word Characters	559
X / XX - Exclude Lines	561
(Column Shift Left	562
) Column Shift Right	563
< Data Shift Left	564
> Data Shift Right	565
[Indent Shift Left	566
] Indent Shift Right	567
Primary Commands	569
Primary Command Notation Conventions	572
Line Control Range Specification	573
Color Selection Criteria Specification	576
Color Change Request Specification	579
ACTION - Request periodic SAVE/SAVE	582
ADD - Add a text line	583
APPEND - Add text to end of line	584
AUTOBKUP - Control Creation of Backups	586
AUTOCAPS - Control AutoCaps Mode	587
AUTONUM - Number lines automatically	589
AUTOSAVE - Control SAVE Action at END	590
BOTTOM - Scroll to Bottom of File	591
BOUNDS - Set Edit Boundaries	592
BROWSE - Open a File for Read-Only	594
CANCEL - Cancel Edit Session	596
CAPS - Set Keyboard CAPS Mode	598
CASE - Set Default String Case Handling	600
CHANGE - Change a Data String	601
CLIP - Open New File Tab with Clipboard Data	606
CLONE - Open an Unnamed Edit Using an Existing File	607
CMD - Execute Another Program or Command	608
COLLATE - Specify Display Character set	609
COLS - Control Visibility of Top Columns Line	610
COMPRESS - Compress Duplicate Strings	611
COPY - Include an External File	614
CREATE - Create an External File	616
CRETRIEV - Conditional Retrieve	618
CUT - Cut Data to the Clipboard	619
DELETE - Delete Selected Lines	622
DIR - Display Folder in File Manager	627
DO - Invoke a KB Command file	628
DROP - Delete Specified Lines in a Tag Group	629

EDIT - Open a File for Editing	630
END - End the Edit Session	631
ENUMWITH - Change Increment for Enumerate Functions	633
EOL - Set End-Of-Line Handling	634
EXCLUDE - Exclude Lines (Edit Mode)	637
EXCLUDE - Exclude Lines (File Manager Mode)	640
EXIT - Terminate SPFLite Session	641
FAVORITE - Add Current File To A Favorite List	643
FF - Find In Files	645
FIND - Find a Character String	647
FIND - Find File Name in File Manager List	654
FLIP - Reverse Exclusion Status of Lines	655
FOLD - Display in Uppercase	658
GLUEWITH - Specify Join String for GLUE operations	659
HELP - Display Online Help	660
HEX - Enable Hex Display of Data	661
HIDE - Hide Excluded Lines	662
HILITE - Control Text Highlighting Options	663
JOIN - Join lines Using Find/Change Strings	664
KEEP - Retain Specified Lines in a Tag Group	671
KEYMAP - Display Keyboard Settings Dialog	672
LC - Lower-Case a Range of Lines	674
LINE - Apply Line Command	675
LOCATE - Scroll to a Specific Line	677
LRECL - Specify Record Length	681
MAKELIST - Create FILELIST	682
MARK - Turn Mark ON or OFF	684
MEDIT - Add a File To a Multi-Edit Session	685
MINLEN - Set Minimum Record Length	686
NDELETE - Delete Lines Where String is Not Found	687
NEXCLUDE - Exclude Where String is Not Found	691
NFIND - Find Where String is Not Found	694
NFLIP - Negative Reverse Exclusion Status of Lines	697
NOTIFY - Set Temporary File Notification Level	700
NREVERT - Negative Revert of User Line to Ordinary Line	701
NSHOW - Show Lines Where String is Not Found	703
NULINE - Negative Mark of User lines	706
NUMBER - Verify and generate sequence numbers	708
NUMTYPE - Enable Numbering and Define Numbering Type	709
OPEN - Open New SPFLite Instance and Edit File	711
OPTIONS - Set Global Editor Options	712
ORDER - Reorder File Line Numbers	713
PAGE - Set PAGE option	714
PASTE - Paste Data from the Clipboard	716
PREPEND - Insert text as start of line(s)	718
PRESERVE - Control Handling of Trailing Blanks	720
PRINT - Send Selected Lines to the Printer	721
PROFILE - Display Current File Profile Variables	725
PTYPE - Enter PowerType Mode	728
QUERY - Display a Profile Setting	730

RCHANGE - Repeat Change	731
RECALL - Recall (Open) a Favorite File List	732
RECFM - Set Record Format	733
REDO - Redo an UNDO Action	734
RELOAD - Reload Current Edit File	735
RENAME - Rename the Current Edit File	736
RENUM - Evenly renumber lines in file	737
REPLACE - Replace a File	739
RESET - Reset (Edit Mode)	741
RESET - Reset (File Manager Mode)	744
RETF - Retrieve Forward Direction	745
RETRIEVE - Retrieve Previous Commands	746
REVERT - Revert User Line to Ordinary Line	747
RFIND - Repeat Previous Find Command	749
RLOC - Repeat Previous Locate Command	750
RLOCFIND - Repeat Previous Find or Locate Command	751
RUN - Execute the current Script	752
SAVE - Save Data and Continue Edit	753
SAVEALL - Save All Current Tabs	754
SAVEAS - Save as New Name and Switch to it	755
SC - Sentence-Case a Range of Lines	757
Scroll Commands - UP / DOWN / LEFT / RIGHT	758
SET - Set a Command Variable	760
SETUNDO - Control Undo Levels	765
SORT - Sort the Edit Data	766
SHOW - Show Lines Where a String is Found	770
SOURCE - Specify Character Encoding	772
SPLIT - Split Lines Using Find/Change Strings	774
START - Set Initial File Position Option	782
STATE - Control Saving of Edit State Information	784
STATS - Turn file STATS ON/OFF	785
SUBARG - Set Default SUBMIT Argument	786
SUBCMD - Set alternate command for SUBMIT	787
SUBMIT - Pass Lines to an External Command File	789
SWAP - Switch to a Selected File Tab	794
TABS - Turn Tabs On or Off	796
TAG - Alter Tag Status of a Range of Lines	797
TC - Title-Case a Range of Lines	801
TOP - Scroll to Top of File	802
UC - Upper-Case a Range of Lines	803
ULINE - Mark User lines	804
UNDO - Undo Changes	806
UNNUMBER - Remove sequence numbers	807
VIEW - View a File in Browse Mode	808
VSAVE - Perform a Virtual Save of Data	810
WDIR - Open Windows Explorer	812
XSUBMIT - Submit an external file	813
XTABS - Control Incoming Tab Characters	814
Appendix	816
Introduction to Keyboard Primitives	816

Index to Keyboard Primitives	818
List of Keyboard Primitives	821
Supplied Fixed Fonts	833
Automatic Colorization Files	843
IBM ISPF Command Support	848
Abbreviations	852

Contents of Article

[Introduction](#)
[SPFLite Evolution](#)
[Version 1.0 thru 4.0](#)
[Version 5.0 thru 6.2](#)
[Version 7.0 thru 7.1](#)
[Version 8.0](#)
[Version 8.1](#)
[Version 8.2](#)
[Version 8.3](#)
[Version 8.4](#)
[Version 8.5](#)
[Version 9.0](#)
[Version 10.0](#)
[A Note about Keyboard Functions](#)
[A Note about Screen Displays](#)
[A Note about System Compatibility](#)
[IBM ISPF Background Information](#)
[Licensing / Distribution](#)

A note about documentation changes

When a Help topic has been updated since the previous full release, the topic icon will have a *** red asterisk** in

 the upper left corner. You will find this indicator in the compiled help (.chm) and HTML versions of the documentation.

Introduction

SPFLite is a Windows text and source-program editor designed for users who are familiar with the mainframe IBM editor known as SPF, ISPF or ISPF/PDF. These original editors function in a significantly different manner than most Windows-based editors, because they are **line-oriented** and **command-driven**. If you've had many years of experience using one editor, changing to another one quite often is disruptive and frustrating.

Functions that one is accustomed to are missing or operate totally differently, and the cry is heard, "Where's my good old ISPF ?!"

I went through this frustration years ago, and after trying out just about every Windows PC Editor I could lay my hands on, I realized none of them satisfied what I wanted. I looked for other ISPF clones, and discovered that there just isn't much out there. The commercial products (e.g. CTC's version) were **way** out of my price range, and secondly looked like they had been programmed by PC/Windows programmers who were basically clueless about what ISPF was really like.

So, as my wife so bluntly put it, "You've been a programmer all your life, create it yourself!" So that's where this came from.

Enter SPFLite.

By the way, before we touch on the history of SPFLite, we'd be remiss if we didn't answer the question most users might be thinking right now, and that is, "Well, an SPF-style editor is great if you are an ex-mainframe developer, but I never was! Why should I care about *your* editor? What's in it for *me*?"

The short answer is, a lot. Besides the fact that a line-oriented editor model is actually a very good one when you are writing programs, SPFLite's editing capabilities are now much better, in many ways, than even the most

recent mainframe releases of ISPF.

Believe it or not, if you have been keeping up with SPFLite's continually improving capabilities, and if you also have opportunity to work on ISPF on a current IBM z/OS system, you may actually find yourself saying the opposite of what George wrote above: "Where's my SPFLite ?!". It's that much better - RH

In addition, the latest versions of SPFLite have added many functions and capabilities that help *bridge the gap* between line-oriented editing and character-oriented word processing. So, what's in it for you? *Lots*.

SPFLite Evolution

I had toyed with ISPF clones from the very early days of personal computers. My first effort was on a 64K (yes, **K**) TRS-80 and looking back it was pretty pathetic. Next came a ported version on the early PCs, sort of usable, but very, very limited.

When I retired, I needed something to keep busy with, so I resurrected my old SPF source and started once again. And SPFLite resulted.

It is simply the Editor portion of ISPF with as many of the features and functions of the editor as I could manage to duplicate. A simple File Manager option was added to provide basic file selection and Delete and Rename functions.

Of course, additions and improvements have been ongoing, and the result is now far past the original 'Lite' designation. Not only does SPFLite support almost all of the IBM SPF commands and facilities, but it has extended these significantly in many areas. Some are extensions in capabilities of existing SPF commands, but many are entirely new, powerful additions to the command set.

Versions 1.0 thru 4.0

Starting as a console-mode application, SPFLite progressed through the conversion to a Windows application, and added multi-tabbed editing, and proper Options/Preferences support so manual editing of INI variables was no longer required. Many ISPF commands and features were implemented.

Version 5.0 thru 6.2

A total keyboard rewrite now allowed almost every keyboard key to be customized. Many new features (Multi-Edit, PowerTyping, Find in Files and Text Highlighting) were added, along with numerous new commands, and keyboard primitives.

Version 7.0 and 7.1

Introduction of a true programmable macro capability. This supports creation of both new Primary commands and Line commands to accomplish custom editing functions. More new features and commands. User lines were introduced along with supporting commands. STATE support switched away from using ADS files to normal files in the SPFLite folder.

Version 8.0

Introduced a significantly revised File Manager, both in screen layout and capabilities.. In addition, a major review and revision of SPFLite internal code was completed. Several other enhancements were made: File specific STATE setting, simplification of Line Command aliases, Dynamic colorization was added, an AUTOCAPS mode was made available when editing programming languages where it is desirable to have language keywords uppercased for readability, optional Keyword hi-lighting on the command line, plus other minor enhancements.

Version 8.1

Several new features were added: Primary command chaining, separation of BROWSE into BROWSE and VIEW commands, a new EXCLUDE command for File Manager to allow filtering of a File List, a new profile option - ACTION - to request automatic SAVE / VSAVE at specified intervals as well as numerous corrections for issues introduced during the V7 to V8 code revision.

Version 8.2

8.2 consisted mainly of some major internal revisions and restructuring to assist in long term maintenance and reliability. A few minor enhancements were included: Improved File Manager 'nesting' so a return to a prior level will re-display the previous scroll and cursor positions; add a new FM option to allow choosing the location of Directory entries within a list . (Top, bottom, or in-line in alphabetical sequence)

Version 8.3

- Add a new CHANGE literal type - **E'xxxx'** which will invoke a user macro to intervene and create the new 'To' literal value. This provides a programmable interface for complex change requirements.
- Add another CHANGE literal type - **M'xxxx'** which provides a powerful character string mapping ability to manipulate the Found string into a result string.
- Add two new AUTO file commands **INCLUDE** and **EXCLUDE** to allow filtering of complete lines. This enables an AUTO file to cause colorization to be only applied against selected line types.
- Many other small error corrections

Version 8.4

- Upgrade thinBasic macro support to the current thinBasic release.
- Revise handling of searching for literals ending with spaces at end of line.
- Revise options for handling Upper/Lower case support
- Many other small error corrections.

Version 8.5

- Support for ISPF-style sequence-numbering of files. See [Working with File Sequence Numbering](#) for more information.
- Enhance the SPF_Parse() macro function for additional flexibility.
- Add a MARKUNIQ option to the SORT command.
- Many bug fixes

Version 9.0

This version was never released. It contained a lot of experimental code which never really worked as expected.

Version 10.0

A total revision of handling screen colors to integrate the specification of screen item colors and the color specifications within the AUTO colorization files. There had always been inconsistencies between the two making color setup awkward and confusing. This change adds additional Hi-light colors and immediate visual feedback of color choices. Color support was also extended to the PRINT output.

The status bar size was increased to allow for a larger font size, and the content and layout of the various status bar boxes can now be controlled by the user. This change also increased the size of the Tab Titles.

Some enhancements to Keyboard handling and a new DO command to improve on keyboard macros..

A couple of other minor additions, see the Change Log (Your Windows Start Menu -> SPFLite -> Change Log).

A Note about Keyboard Functions

As noted above, IBM ISPF is **line-oriented** and **command-driven**, and so is SPFLite. However, a major component of SPFLite - and one of its distinguishing features - is its support for [Keyboard Customization and Keyboard Macros](#) and [Keyboard Primitives](#). A great deal of SPFLite's power, and what makes it so special, derives from this capability. It is **just** as important to understand and use this feature as it is to understand and use SPFLite's array of primary and line commands.

If you choose to use SPFLite as if it were "just a PC version of IBM ISPF" without exploring key-mapped functions, there is nothing "wrong" with that, of course. However, you would be missing out on a large number of highly useful and very cool features that can save you time and allow you to do things you couldn't otherwise do, or

could only do with great difficulty.

Don't miss out! Do yourself a favor, and read up on key mapping and keyboard functions - then try them out. If there's *anything* you don't understand about these functions, and this Help file isn't clear enough, be sure to send us your feedback on the SPFLite Forum. Whether it's a documentation improvement or a feature enhancement that's needed, we do take everyone's feedback into consideration.

Rest assured - the learning curve is well worth the effort.

P.S. For any of you former Tritus SPF users (I am in that group - RH), who might be familiar with the "keyboard primitives" that Tritus supported, you should be aware that the mappable functions in SPFLite are much more powerful. There are loyal Tritus users who thought they'd "never" give up their favorite editor - which was unquestionably a fine product in its day. If you count yourself among such users, you should know that in terms of sheer editing capability, SPFLite is not only every bit as good as Tritus was, it's **better**. SPFLite **is** the successor to Tritus SPF that you have been waiting for!

A Note about Screen Displays

This document shows a number of screen shots to help explain how SPFLite operates. Many will appear with a very small screen size just large enough to contain the lines being discussed. This is done only for convenience in preparing the Help document, and nothing else is implied. You are free to adjust your SPFLite screen size to anything that is convenient for you.

As well, since there are so many screen shots used throughout this document, you will see snapshots created from several different releases of SPFLite. Replacing each and every screen shot with a sample from the current release would be very time-consuming and labor-intensive, and is not usually necessary anyway. Be assured that the purpose of an 'old' screen shot is not lessened because it is from an older release. These are updated as needed if the contents would be misleading. We **do** review every screen shot with every release, and anything that changed in any significant way **is** updated. Don't worry, we've got it covered!

A Note about System Compatibility

SPFLite will operate on Windows 10, Windows 8.0, Windows 8.2, Windows 7, and Windows Vista. Windows XP users must use an older version maintained on the Download page of the SPFLite web site.. As with any other aspect of SPFLite, if you ever run into problems, be sure to send your feedback to the SPFLite Forum so we can look into it.

IBM ISPF Background Information

Because the design of SPFLite is largely based on IBM ISPF, it may be helpful to read the specifications for the original mainframe editor for comparison purposes. Documentation of this mainframe software can be found on the Internet. The most important manual for ISPF is the ISPF Edit and Edit Macros Manual. A recent version of this manual online can be found at:

<http://publibz.boulder.ibm.com/epubs/pdf/ispzem80.pdf>

Licensing / Distribution

SPFLite is distributed as Shareware and all copies and versions are **fully functional** and will remain fully functional even after the trial period has expired. The only difference after the trial period, if you have not registered the program, is that it will randomly 'nag' every few uses as it terminates. I don't like 'nag-ware' either, so I've tried to keep it at a minimum. But you are basically on your honor; if you use it often enough for the nagging to bother you, you also know how to stop the nagging.

Support and Contacts

The latest release of SPFLite may be obtained at any time from the SPFLite web site:

www.SPFLite.com

It can also be found on many other download sites such as:

download.cnet.com

www.BrotherSoft.com

www.SoftPedia.com

but these sites **do not** always have the latest release. After a new release, there is usually a delay before these other download sites get refreshed. The www.SPFLite.com site will **always** have the latest correct version available.

For other specific questions or support regarding SPFLite, you can send e-mail to:

Support@SPFLite.com

There is also an SPFLite Forum where you may pose questions, make suggestions etc. It is located at:

<http://spflite.freeforums.net/>

Users are encouraged to use this SPFLite Help document as their primary resource in resolving questions. Extensive efforts were made to thoroughly document SPFLite's features, and most questions you are likely to ask may already find their answer herein.

We do test each release as thoroughly as we can, but because software is written by mere mortals, you may find a bug we overlooked, or maybe something just doesn't work quite the way you'd like, or perhaps you wish SPFLite had some feature to make your work easier. Feel free to contact us and let us know what's on your mind. We may be able to suggest workarounds or editing techniques that can help get your work done.

Major Features of SPFLite

Contents of Article

[Major Features of SPFLite](#)

[Summary of Primary Commands](#)

[Summary of Line Commands](#)

Major Features of SPFLite

- Full 32-bit Windows application.
- Standard text-mode Windows clipboard support along with multiple Named Private Clipboards.
- User-customizable screen. Specify your own desired Font and Pitch selection (fixed-width fonts only) as well as custom colors for the various screen components. The screen can also be resized at any time with the mouse, using standard Windows drag arrows.
- Support for national keyboards and alternate layouts (using Windows Control Panel / Regional and Language Options).
- UNDO/REDO support, with a user-specifiable number of levels.
- BROWSE support provides all EDIT capabilities but in a read-only mode.
- Full keyboard mapping and mouse-button mapping, as well as keyboard macros and keyboard recording.
- A programmable macro language to allow automating repetitive functions as well as the creation of new, custom primary and line commands.
- Automatic colorization files can be used to colorize keywords and syntax in programs and scripts.
- HEX-mode editing.
- Regular Expression search strings supported in all Primary commands using search strings.
- Tabbed Edit interface to support multiple edit sessions.
- Drag and Drop support. Files can simply be dropped on an active SPFLite window to open them.
- Integrated File Manager for file selection and basic delete and rename functions.
- EBCDIC and Basic Unicode file editing; EBCDIC code page can be user-customized, or use the 1252/1140 default.
- Print Screen options for output to the clipboard, to a default printer or to a log file, as well as formatted printing support.
- Multi-Edit mode is a new, powerful facility that allows editing several files together in single edit tab. For instance, one CHANGE can alter text in multiple files at the same time, in a single command.
- Power Typing mode allows simultaneous editing of multiple lines (possibly scattered) in parallel (called "column mode" in other editors).
- A configurable SUBMIT command can send jobs to external processes, such as the Hercules emulator.
- Highlight text support (i.e. like yellow hi-lighter) to mark important text areas. Maintained across Edit/Browse sessions.
- Many new primary and line commands have been added that extend the IBM ISPF command set, as well as powerful new extensions to many of the original ISPF commands.

Summary of Primary Commands

The following primary commands are implemented. See the Appendix for the [Abbreviations](#) that are supported.

ACTION	Request automatic SAVE/VSAVE be performed.
ADD	Add a text line
APPEND	Add text to the end of lines
AUTOBKUP	Control backup creation
AUTOCAPS	Control auto-capitalization of language keywords
AUTONUM	Number lines automatically
AUTOSAVE	Control automatic file save defaults
BOTTOM	Scroll to bottom of file
BOUNDS	Set edit boundaries
BROWSE	Open a file for browse (read-only) access, no changes allowed
CANCEL	Cancel edit session without file save
CAPS	Set keyboard CAPS mode
CASE	Control default literal case handling
CHANGE	Change a data string
CLIP	Open a new tab using clipboard data
CMD	Execute another Program or Command
CLONE	Open an un-named edit using an existing file
COLLATE	Specify display/sort collating sequence
COLS	Control visibility of top Columns line
COMPRESS	Compress duplicate strings
COPY	Include an external file
CREATE	Create an external file
CRETRIEV	Cursor/Retrieve
CUT	Cut data to the clipboard
DELETE	Delete selected lines
DIR	Display folder containing this file in File Manager
DO	Execute a keyboard command file
DOWN	Scroll downward in the data
DROP	Delete selected lines in a Tag group
EDIT	Open a file for editing
END	End the edit session
ENUMWITH	Change Enumerate increment value
EOL	Alter end-of-line mode
EXCLUDE	Exclude lines from the display (Edit mode)
EXCLUDE	Exclude files from the display (File Manager Mode)
EXIT	Terminate SPFLite session
FAVORITE	Add current file to a Favorite list
FF	Edit/Browse Find command alias FF
FF	File Manager Find in Files command FF
FIND	Find a character string
FLIP	Reverse X and NX lines
FOLD	Display text in Uppercase only
GLUEWITH	Specify join string for Glue operations

JOIN	Selectively join lines together using find/change strings
HELP	Display the Help file
HEX	Set HEX display mode ON or OFF
HIDE	Hide excluded lines
HILITE	Control text highlighting options
KEEP	Delete specific lines in a TAG group
KEYMAP	Display keyboard settings dialog
LC	Lower-case a range of lines
LEFT	Scroll leftward in the data
LOCATE	Scroll the display to a specified line
LRECL	Specify record length
MAKELIST	Create FILELIST from display in File Manager
MARK	Turn Mark lines ON or OFF
MEDIT	Add a file to a Multi-Edit session
MINLEN	Set minimum record length
NDELETE	Delete lines where string is not found
NFIND	Find lines where string is not found
NFLIP	Reverse X and NX lines where string is not found
NEXCLUDE	Exclude lines where string is not found
NONUMBER	Turn Number mode off
NREVERT	Revert User line status where string not found
NSHOW	Show (unexclude) lines where string is not found
NULINE	Mark User line status where string not found
NUMBER	Verify and generate valid sequence numbering
NUMTYPE	Define the numbering 'style' to use
OPEN	Edit another file in a new session
OPTIONS	Set editor global options
ORDER	Reorder file line numbers
PAGE	Set Profile PAGE mode ON or OFF
PASTE	Paste data from the clipboard
PREPEND	Add text to the beginning of line(s)
PRESERVE	Control handling of trailing blanks
PRINT	Print selected lines to the printer
PROFILE	Display current file profile values
PTYPE	Enter PowerType mode
QUERY	Display a single Profile setting
RCHANGE	Repeat change
RECALL	Recall (Open) a favorite File list
RECFM	Set record format
REDO	Redo (back out) a prior UNDO action
RELOAD	Reload current edit file
RENAME	Rename the current edit file
RENUM	Evenly renumber lines in file
REPLACE	Replace a file
RESET	Reset (Edit mode)
RESET	Reset (File Manager mode)
RETF	Recall commands in a forward direction

RETRIEVE	Recall previous commands
REVERT	Revert User line status
RFIND	Repeat the find command
RIGHT	Scroll rightward in the data
RLOC	Repeat last LOCATE command
RLOCFIND	Repeat most recent FIND or LOCATE command
RUN	Directly execute the current Edit script
SAVE	Save data and continue edit
SAVEALL	Save all current tabs
SAVEAS	Save data as a new file and switch to it
SC	Sentence-case a range of lines
SET	Set a command variable
SETUNDO	Control UNDO levels
SHOW	Show (unexclude) lines where a string is found
SORT	Sort the edit data
SOURCE	Specify character encoding
SPLIT	Selectively split lines apart using find/change strings
START	Set initial file position option.
STATE	Control edit state saving
SUBARG	Set default SUBMIT arguments
SUBCMD	Set alternate command for SUBMIT
SUBMIT	Pass lines to an external command file
SWAP	Switch to Previous or Next Tab
TABS	Turn TABS On or Off
TAG	Alter TAG status of a selection of lines
TC	Title-case a range of lines
TOP	Scroll to the top of the file
UC	Upper-case a range of lines
ULINE	Mark User line status
UNDO	Undo changes
UNNUMBER	Remove sequence numbers
UP	Scroll upward in the data
VIEW	View a file in read-only mode
VSAVE	Do a virtual save
WDIR	Open Windows Explorer for this file folder.
XSUBMIT	Submit an external file
XTABS	Control handling of incoming tabs

Summary of Line Commands

<u>A</u>	After destination
<u>AA</u>	After block
<u>B</u>	Before destination
<u>BB</u>	Before block
<u>BNDS</u>	Display BOUNDS line
<u>C</u>	Copy line(s)
<u>CC</u>	Copy a block
<u>COLS</u>	Display Columns line
<u>D</u>	Delete line(s)
<u>DD</u>	Delete a block
<u>E</u>	Display first lines in excluded region
<u>G</u>	Glue lines together
<u>GG</u>	Glue lines together in block
<u>H</u>	Here-destination lines
<u>HH</u>	Here-destination block
<u>!</u>	Insert temporary new line(s)
<u>J</u>	Join lines together
<u>JJ</u>	Join lines together in block
<u>L</u>	Display last lines in excluded region
<u>LC</u>	Lower-case lines
<u>LCC</u>	Lower-case lines in block
<u>M</u>	Move lines
<u>MM</u>	Move lines in block
<u>MARK</u>	Set column markers
<u>MASK</u>	Set the Insert line model
<u>MD</u>	Make data lines
<u>MN</u>	Make NOTE lines
<u>N</u>	Insert permanent new line(s)
<u>NOTE / xNOTE</u>	Insert NOTE or xNOTE lines
<u>O</u>	Overlay lines
<u>OO</u>	Overlay lines in block
<u>OR</u>	Overlay-Replace lines
<u>ORR</u>	Overlay-Replace lines in block
<u>PL</u>	Pad lines
<u>PLL</u>	Pad lines in block
<u>R</u>	Repeat lines
<u>RR</u>	Repeat lines in block
<u>S</u>	Show lines
<u>SS</u>	Show lines in block
<u>SC</u>	Sentence-case lines
<u>SCC</u>	Sentence-case lines in block
<u>SI</u>	Show indentation
<u>T</u>	Select text lines
<u>TT</u>	Select text lines in block
<u>TABS</u>	Display Tabs line

<u>TB</u>	Text Break a line
<u>TBB</u>	Text Break lines in block
<u>TC</u>	Title-case lines
<u>TCC</u>	Title-case lines in block
<u>TF</u>	Text-flow a paragraph
<u>TFF</u>	Text-flow a block of paragraphs
<u>TG</u>	Text glue lines together
<u>TGG</u>	Text glue a block together
<u>TJ</u>	Text join lines together
<u>TJJ</u>	Text join a block together
<u>TL</u>	Trim lines
<u>TLL</u>	Trim lines in block
<u>TM</u>	Set Text Margin in a paragraph
<u>TMM</u>	Set Text Margin in a block of paragraphs
<u>TR</u>	Truncate lines
<u>TRR</u>	Truncate lines in block
<u>TS</u>	Text split a line
<u>TU</u>	Toggle the User-Line status of a line
<u>TX</u>	Toggle the excluded status of a line
<u>UC</u>	Upper-case lines
<u>UCC</u>	Upper-case lines in block
<u>U</u>	Mark User lines
<u>UU</u>	Mark User lines in block
<u>V</u>	Revoke User line status
<u>VV</u>	Revoke User line status in block
<u>W</u>	Swap lines
<u>WW</u>	Swap lines in block
<u>WORD</u>	Display valid WORD characters
<u>X</u>	Exclude lines
<u>XX</u>	Exclude lines in block
<u>{</u>	Column shift left
<u>{{</u>	Column shift left in block
<u>}</u>	Column shift right
<u>}}</u>	Column shift right in block
<u>≤</u>	Data shift left
<u>≤≤</u>	Data shift left in block
<u>≥</u>	Data shift right
<u>≥≥</u>	Data shift right in block
<u>[</u>	Indent shift left
<u>[[</u>	Indent shift in block
<u>]</u>	Indent shift right
<u>]]</u>	Indent shift right in block

Installing SPFLite

Contents of Article

[Introduction](#)
[Installing Updated Versions](#)
[Uninstall Considerations](#)
[Initial Execution of SPFLite](#)
[Installation Wrap-up](#)

Introduction

SPFLite is distributed as a standard compressed ZIP file. This file should be uncompressed (using standard UnZip tools, or Windows Explorer) to create the installer executable named SPFLiteSetup.EXE. Next, double-click the EXE file to start the install. Follow the install prompts to complete the installation.

Installing Updated Versions

When a new version of SPFLite is installed, there is **no need** to uninstall the previous version, or save your configuration files. The new version will install "Over the Top" of the existing version and retain all your current settings.

Uninstall Considerations

An uninstall icon is created during the SPFLite install. Double click this icon to remove the program. You can also use the standard Windows Add/Remove program support in Control Panel. Just select SPFLite and click on Remove.

Initial Execution of SPFLite

If this is the very first time SPFLite is executed on your system, you will be presented with some customization options to choose from.

A common problem for new users is to make wrong assumptions about which keys are mapped to the ISPF equivalents of ENTER and NEWLINE.

This can happen if you failed to read, or just forgot, the "[Default Key Definitions](#)" in the Help file, or perhaps the way defaults for ENTER and NEWLINE was not how you expected. Either way, you may end up not knowing where ENTER is. Since primary commands will not operate until the ENTER key is pressed, it would be a serious problem if you couldn't find it.

To avoid that, this Pop-up asks you to choose one of three common mappings to use. Regardless of which you choose, you can **always** change these later using the KEYMAP facility.



At this point, choose whichever of the three options presented that seems closest to your preference. You can alter this later to whatever you desire, using the [KEYMAP](#) command. If you are new SPFLite user, you may wish to make a note of which choice you made.

Installation Wrap-up

Once you have made this selection, you will then be presented with the SPFLite Options dialog so that you can set other program options. If your feeling is "I don't have a clue what to customize" then simply press the 'Done' button. You can alter any of these options later to whatever you desire, using the [OPTIONS](#) command, once you become more familiar..

Note: If you ever **did** run into a problem with the definition of the Enter key, and this is preventing you from using SPFLite properly, you can launch SPFLite and directly start the KEYMAP dialog to resolve this problem. See [Starting and Ending SPFLite](#) for more information.

Creating a Portable Version

Contents of Article

[Introduction](#)

[Retaining Customization](#)

[Using STATE Information on Removable Media](#)

Introduction

SPFLite can be copied to removable media (such as a USB flash drive), complete with its customized environment for use on systems other than the one on which it was originally installed.

If you plan to do this, complete all customization activities **before** performing the following steps.

If you are a registered SPFLite user, also do this **after** the registration is complete.

- Copy the entire SPFLite program files folder from C:\Program Files (x86)\SPFLite to a folder on the removable media
- Inside the folder you created above, create a \Config folder.
- If you are not sure where the SPFLite customization files are stored:
 - Start SPFLite
 - Enter **OPTIONS** (can be abbreviated to **OPTION** or **OPT**)
 - The correct location will be shown at the bottom of the Options dialog box.
- Using the directory located above, copy the entire contents of this directory, including its subdirectories to the \Config folder you created in the 2nd step.
- Your portable version is now complete. When executed from the removable media, all your customization will be retained.

Retaining Customization

If you copy SPFLite from the removable media to another system's hard drive and execute it from that hard drive, SPFLite will NOT retain your customization.

Using STATE Information on Removable Media

Since STATE information is now stored as ordinary data files, under the STATE folder of SPFLite. Your removable media can have any file system you prefer, either NTFS or any type of FAT file system supported by your version of Windows.

See [Saving the Edit STATE](#) for more information.

Differences between SPFLite and IBM ISPF

Contents of Article

[Introduction](#)
[Extensions to many of the standard commands](#)
[New Primary commands](#)
[New line commands](#)
[Additional line command extensions](#)

Introduction

SPFLite differs from mainframe ISPF in several areas, most noticeably in the keyboard handler. You have more flexibility with SPFLite in customizing the keyboard than in ISPF. You can find a full description in "[Keyboard Customization](#)". Even before you customize it, the default keyboard setup for SPFLite is probably different from what you are used to. The defaults are outlined in "[Default Key Definitions](#)".

Note: Some ISPF commands have not been implemented. See [IBM ISPF Command Support](#) for more information on which commands have and have not been implemented, and work-arounds some of these.

You can define keyboard macros and launch them from any keys of your choosing. This is described under "[Keyboard Macros](#)".

As mentioned in the [Welcome to SPFLite](#), the [KEYMAP](#) facility and mappable keyboard functions play an important role in providing many of the powerful features of SPFLite. IBM ISPF did not give as much emphasis to this, because 3270 keyboards only allowed for mapping of PF keys, whereas SPFLite can map almost anything. The time you spend learning about this facility will be well worth the effort.

SPFLite does not support the IBM ISPF feature of "3270 split screen mode" and the associated legacy **SPLIT** command. With tabbed editing and the ability to open multiple instances of SPFLite, as well as the Multi-Edit feature to edit several files simultaneously in the same edit window, IBM's 3270 split screen mode is not really needed.

Note: The SPFLite command **SWAP PRIOR** will alternate the 'focus' of the editor between the last two edit tabs you have referenced. The F9 key has a standard mapping of **SWAP**, so if you wanted it to be **SWAP PRIOR** you would have to adjust its definition. Some users may prefer mapping the **SWAP PRIOR** command to Ctrl-Tab, since that key is often used for a similar function in other Windows programs. If you use **SWAP PRIOR** in this way, it will achieve much of what a mainframe ISPF user would do when editing two files in a 3270 split screen session and swapping between them with F9.

The ISPF standard of retaining a command on the command line after completion is supported, by prefixing the command with an & character.

SPFLite utilizes the mouse where possible, to enhance your productivity. See "[Use of the Mouse](#)" for details.

Besides including nearly all standard ISPF commands and functions (See [IBM ISPF Command Support](#) for information on what commands are supported). SPFLite has added editing features not found in mainframe ISPF or in PC-based SPF editors of the past. These features include the following.

Message Handling

ISPF supports a multi-level message handling ability. When a message is issued, you can press HELP for a more detailed message and a 2nd time to go directly to a specific Help topic related to the error.

SPFLite does not provide this extensive level of support. When messages are issued internally by various command processors, SPFLite tracks the severity of the various messages and will ultimately issue the

message flagged as being 'most severe'

Generally, this works quite well and there is no problem determining the error condition. However sometimes, especially when several commands are issued together in one interaction, it may not be entirely clear what has gone wrong, and being able to review all issued messages would enable a quick resolution.

Whenever multiple internal messages have been issued, and SPFLite has chosen the 'most severe' one to display, the message will be prefixed with a **+** character, to indicate that multiple messages have occurred.

If a **HELP** command is immediately issued at this point (while the **+** prefixed message is still displayed, SPFLite will open a pop-up list of all issued messages as they were issued for your review.

Extensions to many of the standard commands

- Powerful line range specification criteria. You can select lines by:
 - Simple line range (From - To)
 - Conditional based on a label (such as, all lines < a specified line)
 - Conditional based on two labels (such as, all lines >= lineA and <= lineB)
 - Split ranges (such as, all lines < lineA OR > lineB)
 - Scattered (non-contiguous) lines marked by a common Tag name
- **MX** option requesting the lines processed be excluded following the command
- **DX** option to suppress the normal 'popping out' of excluded lines by commands like **FIND** and **CHANGE**
- Ability to mark a range of lines via **C/CC** and **M/MM** line commands for use on a wide variety of primary commands
- Ability to segregate lines into two groups: User Lines (also called U lines) and non-User Lines (also called V lines, or NU lines)

New Primary commands

- **ADD** to insert a string as a new line
- **APPEND** to add a string to each line
- **AUTOCAPS** to control capitalization of language keywords
- **CASE** to control default case handling of literals
- **CLIP** to directly edit the contents of the clipboard
- **CLONE** to create an unnamed working copy of a file
- **COMPRESS** to eliminate duplicate character strings
- **DIR** to open a file display of the folder containing the current file
- **DROP** to assist in manipulating tag groups
- **ENUMWITH** specify increment to use with Enum functions
- **EOL** to accommodate different End of Line delimiters
- **EXIT** to close all active tabs
- **FAVORITE** to add current file to a Favorite FILELIST group
- **FF** (Find in File) searches a list of files for a given string
- **FIND** in the File Manager locates file **names** containing a given string
- **GLUEWITH** specify insert character to be used with Glue functions
- **JOIN** selectively joins lines using find/change strings
- **KEEP** to assist in manipulating tag groups
- **LINE** command to apply a normal Line Command to a range of selected lines.
- **LC/UC/SC/TC** to perform text case conversions
- **MARK** controls the visibility of previously set MARK lines
- **MAKELIST** creates user-defined FILELIST groups from File Manager
- **MEDIT** allows simultaneous editing of multiple files in a single edit session
- **MINLEN** sets the minimum logical record length of a given file type
- **NFIND** locates lines which do **not** contain a specified string

- **NFLIP** inverts the exclude status of lines which do **not** contain a specified string
- **NEXCLUDE** excludes lines which do **not** contain a specified string
- **PAGE** to set Page display settings for SYSPUT type files
- **PREPEND** insert a string at the beginning of each line
- **PTYPE** begins Power Typing ("column") mode for simultaneous editing of multiple lines in parallel
- **RECALL** displays a FILELIST group
- **REDO** reverses a prior UNDO action
- **RELOAD** to restart an edit session from the current contents on disk
- **RENAME** allows renaming of the current edit file
- **RETF** perform a RETRIEVE in forward order
- **SAVEALL** will SAVE all open edit sessions
- **SET** assigns or queries the value of a SET variable
- **SPLIT** selectively splits lines using find/change strings
- **START** controls where a file is initially positioned when opened
- **STATE** controls whether persistent edit STATE information is maintained
- **SUBARG** specify a Profile unique string for SUBMIT use
- **SUBMIT** enables jobs to be submitted to external processes such as Hercules
- **TAG** to define and modify line tags
- **ULINE, NULINE, REVERT, NREVERT** manage the User-Line status of lines
- **VSAVE** performs a Virtual SAVE of an edit file, leaving the original version on disk untouched

New line commands

- **AA** adds an After-group to allow copying/moving lines multiple times, after every n'th line
- **BB** adds a Before-group to allow copying/moving lines multiple times, before every n'th line
- **G** and **J** are used to Glue and Join lines together
- **H/HH** define a Here-destination, which is similar to **A/B** except the marked lines are **replaced** by the copied/moved lines
- **MARK** defines the placement of vertical 'marker' lines that help in editing column-aligned data
- **MD** (Make Data) will convert special lines into data lines.
- **MN** (Make Note) will convert special and data lines into NOTE lines
- **N** inserts new, permanent blank lines, unlike the **I** command which inserts temporary blank lines
- **NOTE** inserts NOTE lines
- **OR** (Overlay-Replace) forcibly overlays data without the conflicts that can occur with regular Overlay
- **PL** (Pad to Length) pads data lines with blanks, to a specified line length
- **S/SS** (Show line/block) will unexclude lines, and is the exact opposite of the **X/XX** command
- **SC** and **TC** change text to Sentence Case or Title Case, similar to **UC** and **LC** commands
- **T/TT** will select text, similar to using the mouse or shift-arrows, but across multiple lines
- **TFF** performs **TF** formatting across multiple paragraphs in a single interaction
- **TG/TJ** Glue and Join lines in "text" mode, and are similar to **G/J**
- **TL** and **TR** adds a Trim and Truncate line ability to remove trailing blanks or reduce line size
- **TM/TMM** sets Text Margins in a manner similar to but simpler than **TF** does
- **TB/TBB** performs a Text Break (like Text Split) inserting permanent instead of temporary blank lines in between
- **TU/TUU** toggles the User-Line state of lines
- **TX/TXX** toggles the excluded state of lines
- **U/UU** sets lines to be User Lines
- **V/VV** sets lines to be ordinary V lines; that is, non-User Lines
- **W/WW** defines one end of a line Swap operation (with an **M/MM** block at the other end)
- **WORD** supports user customization of what are considered valid word characters
- **[and]** perform *Indent Shifts* based on a number of columns set in the Global Options screen

Additional line command extensions

- Forward modifier **/** and Backward modifier **** for line commands. These allow quick specification of 'all lines from here to bottom' and 'all lines from here to top' to line commands which accept a line count

parameter. The forward modifier is equivalent to entering a count of 99999 to mean the rest of the file. (There is no ISPF equivalent of 'all lines from here to top'.)

- The – Post-Exclude and + Post-Unexclude modifiers. These extensions allow you to request that the lines processed by the line command groups be excluded or unexcluded following the command.
- Retain line commands. If an extension of **&** is added to a line command, it is a request to retain the line command on the line following completion of the line command. This allows you to reuse a line command over and over, until it is manually erased or you issue a **RESET** primary command. This is similar but not identical to ISPF "K" command usage, such as **AK**.

See [Extended Line Command Modifiers](#) for more information.

Use of the Mouse

Contents of Article

- [Introduction](#)
- [Standard Windows Dialogs](#)
- [Scrolling](#)
- [Cursor Positioning](#)
- [Keyboard Key Simulation](#)
- [Tab Switching](#)
- [Tab Closing](#)
- [Text Selection](#)

Introduction

The original ISPF 3270 terminals had no mouse and SPFLite was initially designed to operate in the same manner, using only the keyboard. But because SPFLite **is** a Windows program and has access to the mouse, SPFLite provides mouse support to enhance user productivity in the following areas:

Standard Windows Dialogs

At various times SPFLite will utilize standard Windows dialogs to prompt for information. When this is done, these dialogs allow you to use the mouse to make selections and button choices. Examples of this are:

- File selection menus.
- Font selection menus
- Color selection menus
- Printer selection menus

Scrolling

If scrolling is activated (see ["Options - General"](#)) the mouse wheel may be used to scroll the text window in an edit session, or a list of files in the File Manager. The number of lines scrolled by each mouse-wheel click is set in Options.

Scrolling is normally done vertically. Holding the **Shift** key while using the mouse wheel causes horizontal scrolling. (Think: "shift = sideways")

Holding the **Ctrl** key while using the mouse wheel will cause a 4 X speedup of the scrolling action when you want to scroll long distances within a large text file. This is called Turbo Motion, or Turbo Mode scrolling.

If you want to do horizontal scrolling in Turbo Mode, you can hold the **Ctrl** and **Shift** keys at the same time while using the mouse wheel.

Cursor Positioning

During file editing, the left mouse button can be clicked anywhere within the screen area to move the cursor quickly to that location, and will then **optionally** issue an action mapped to an SPFLite keyboard key, if that action is configured in the [KEYMAP](#) settings.

Keyboard Key Simulation

As well as positioning the cursor on the screen, the left, middle and right mouse buttons can be used to (after the positioning) simulate the entry of a single keyboard key. Note that with keyboard mapping, this can launch

powerful commands and macros. See "[Keyboard Customization and Keyboard Macros](#)" for details.

Tab Switching

Similar to the [SWAP](#) command for tab switching, you can left-click on any file tab, or File Manager tab, to directly switch the active edit session or activity to that tab.

Tab Closing

When a file tab is clicked with the right mouse button, it closes the file in the same way that the [END](#) command does. The PROFILE option [AUTOSAVE](#) is handled in the same way as is done for the [END](#) command. A right mouse click on the File Manager tab displays the next higher directory level.

Text Selection

SPFLite provides a number of features where the mouse is used in text-editing operations. You can find more information in the article [Word Processing Support](#).

During Edit, the left mouse button can be held down and dragged over text to select it as a block. The selected text will be displayed on the screen in inverse video to highlight the area you have selected. Note that the default mode (termed '1-Directional' or '1-Dimensional') will ignore any vertical mouse movements which would otherwise go outside the line on which the selection started. This restricts you to selecting from a single text line, which is the most common usage and is designed to prevent unintended multi-line selects.

If you wish to select a multi-line block of text, hold down any Shift, Ctrl or Alt key and then select with the mouse in any direction you wish to highlight a block. This mode is referred to as a '2-Directional' or '2-Dimensional' select. You can also choose to make multi-line selection the default to avoid the need for holding down an additional key. Your choice.

If text is double-clicked while the mouse is not moving, a span of characters will be selected that are delimited by the current WORD characters and/or the end of the line. This operation performs a 'word select'.

Once selected, the highlighted text can be placed on the clipboard using the keyboard [\(Copy\)](#) command (normally assigned to Ctrl-C). If you have selected the incorrect data, simply release the left mouse button and start over again.

In addition, keyboard functions exist for [\(Cut\)](#), [\(Lift\)](#), [\(Paste\)](#), [\(BlockPaste\)](#) and [\(CopyPaste\)](#), any or all of which could be assigned as action commands to the mouse buttons. [\(CopyPaste\)](#) provides the functionality of the Windows QuickEdit feature, which will copy text if any is highlighted, and will paste text if there is no active highlighting. [\(Paste\)](#) now supports the block mode previously performed by [\(BlockPaste\)](#), so that the [\(BlockPaste\)](#) command isn't really needed any more, but is still available for backward compatibility.

The selected data area is restricted to what is visible on the current screen. You may not scroll the screen while selecting text with the mouse.

If you want to select more data than is visible vertically, you can use the [Exclude](#) command [\(X\)](#) to conceal lines you are not interested in, or use keyboard text selection (shifted arrow keys) which will allow screen scrolling.

Any keyboard function like [\(Upper\)](#) that operates on a selected-text area, will now also operate on a single character if the cursor is within the data area of a line, even if the data at that cursor position not highlighted. This makes it easier to do things like capitalize a single letter, or any other similar keyboard function.

Note: You may also use the [T/TT](#) line command to select text across multiple lines, which is useful when the number of lines involved is large. See [T / TT - Select Text Lines](#) for more information.

Default Key Definitions

Contents of Article

[Introduction](#)

[SPFLite standard key mapping defaults](#)

[Standard F1 through F12 key mappings](#)

[Standard Keypad key mappings](#)

[Additional installation-supplied key mappings](#)

Introduction

As described under "[Keyboard Customization](#)" you can map the physical keyboard keys to any logical Primitive functions you desire.

During the first execution of SPFLite following install, you are presented with a popup display and asked to choose between three different key assignments for the ENTER and NEWLINE keys. You must choose one of the options. If none suit your preference, you must still choose one to start with; you can immediately change these settings with the **KEYMAP** primary command. The initial assignment here is done to prevent SPFLite from setting defaults that you are not aware of. Since the particular choice you make is important, you may wish to make a note of it somewhere.

The popup display looks like this:



Your choice above will then be included with the standard key defaults.

SPFLite standard key mapping defaults

SPFLite is installed with a standard set of predefined keys to get you started. Many users choose to immediately alter some of these to their own preferences. The predefined defaults reflect common usage in mainframe ISPF, PC-based ISPF emulators and text-mode Windows editors. Most users will find these defaults a useful place to begin.

Why doesn't SPFLite just map every possible function to a standard set of keys for you, instead of requiring you to make mapping choices for all these functions? First, you may not need or be interested in using every last function, but just the ones that are important to your work. Second, you may have experience with other editors, and you probably would rather have **your** keys do what **you** want them to, instead of having SPFLite make those choices.

As well, your physical keyboard may dictate some choices. a full 104 key keyboard has many more keys available over a laptop keyboard. What works well on a laptop would simply be ignoring the 17 keys available in the keypad of the 104 key keyboard, not taking advantage of those keys wold be a real shame.

Throughout this Help, you will find **suggested** mappings for various commands and functions, which have been found to be useful. You are free to use these suggested mappings, or pick your own.

The following defaults are installed

- All alphabetic and numeric keys operate as normal.
- Insert / Delete / Home / Tab and Backtab (Shift-Tab), and Backspace operate as normal
- Page Up / Page Down keys by are mapped to generate the UP and DOWN commands.

Standard F1 through F12 key mappings

- **F1** Help
- **F2** Reset
- **F3** End
- **F4** Unassigned
- **F5** RFind
- **F6** RChange
- **F7** Up
- **F8** Down
- **F9** Swap
- **F10** Left
- **F11** Right
- **F12** CRetriev

Standard Keypad key mappings

- **KP-1** End
- **KP-2** Down Arrow
- **KP-3** PageDn
- **KP-4** Left Arrow
- **KP-5** Null
- **KP-6** Right Arrow
- **KP-7** Home
- **KP-8** Up Arrow
- **KP-9** PageUp
- **KP-0** Insert
- **KP-. (Period)** Delete
- **KP-Enter** Enter

Additional installation-supplied key mappings

- **ESC** Erase EOL
- **PrtScr** Print screen to Clipboard
- **S-PrtScr** Print screen to printer
- **C-PrtScr** Print text portion of screen only to Clipboard
- **A-PrtScr** Print screen to SPFLite log file

- **ScrlLock** Toggle keyboard recording mode
- **C-X** Cut selected text to the Clipboard
- **C-C** Copy selected text to Clipboard
- **C-V** Paste text at cursor location; this will also perform block-paste operations
- **End** Cursor to end of line
- **S-End** Mark text to end of line
- **S-Arrows** Mark text in arrow direction
- **C-Left** Move cursor left one word
- **C-Right** Move cursor right one word

Customizing SPFLite

SPFLite provides a large number of option settings to allow you to customize it to your personal preferences. An initial set of default values is installed with the program. You should review these initial option settings to see if they meet your requirements.

If this is your first time using SPFLite, you will automatically be presented with the SPFLite General Options dialog to allow you to set the initial options you prefer.

At any time afterwards, you can enter the **OPTIONS** command to display the General Options dialog.

SPFLite options are grouped into three categories, each accessible by a primary command. Review each tab's contents, and when complete, click on the **Done** button at the bottom.

Organization of Options

User options are stored in the following categories:

General Options

The **OPTIONS** command will display a tabbed dialog, where preferences for general or global options are maintained. These options are related to the general operation of SPFLite, and not with the editing of any particular file type.

Profile Options

These are options which are linked to a particular file type. Options such as automatic colorization support, tab settings and delimiter characters are controlled with a **PROFILE**.

When your file has a file name extension, its file type is the same as the file name extension. So, a file called MyFile.txt has a file type of **TXT**, and that is the name of its PROFILE. When your file does not have a file name extension, you can use the name of the file's parent directory as its assumed profile name, if you select the General Option "Use DIR name as PROFILE when no file extension".

When you wish to see the PROFILE options for a given file you are editing, enter the **PROFILE** command with no options. This will display several special lines in your edit session, like **=PROF>**. The [QUERY](#) command can also be used for a quick display of any single Profile setting. You can alter profile options using keyword commands such as **CAPS** and **AUTOSAVE**. If you want to edit a profile other than the one for your current edit file, you can use the **PROFILE EDIT name** command.

Keyboard and Mouse Mapping

Keyboard mapping definitions are accessed with the **KEYMAP** command, which will display a KEYMAP dialog. This allows you to customize the keyboard operation by defining the SPFLite functions and keyboard macros that are assigned to the keys you desire.

SET Options

It is also possible to control some optional SPFLite operating features using **SET** commands. The **SET** names involved are:

SET ALIAS.shortname = longname

SET AUTOFAV.filepath = namedFavorites_name

Options - General

Index of General Options

[Audible BEEP on Errors ?](#)
[Visual BEEP on Errors ?](#)
[Only 1 SPFLite running ?](#)
[Re-Open last file\(s\) at Start ?](#)
[Delete to Recycle Bin ?](#)
[Warn on modified View file ?](#)
[Use DIR name as PROFILE when no File extension ?](#)
[Use WORD as the default for FIND/CHANGE commands ?](#)
[Allow 2-D mouse selection without Shift/Ctrl/Alt ?](#)
[Only English letters A-Z and a-z are considered alphabetic](#)
[Default RESET will Revert User Line Status](#)
[Minimum command length for Retrieve ?](#)
[Command Separator Char.](#)
[Use fast renum if # lines > this value](#)
[Default # columns for Data Shift \(Min 1\)](#)
[Number of Columns/Lines per Auto-Scroll](#)
[Notify tabs on external file change](#)
[Line Commands Repeat Limit](#)
[Normal characters for P'.' Picture Literals](#)
[Display Char. for invalid characters](#)
[Cancel and Done Buttons](#)

Audible BEEP on Errors? Only 1 SPFLite running? Visual BEEP on Errors? Re-Open last file(s) at start? Delete to Recycle Bin? Warn on modified View file? Use DIR name AS PROFILE when no File extension? Display splash screen on startup? Use WORD as the default for FIND/CHANGE commands? Allow 2-D mouse selection without Shift/Ctrl/Alt? Only English letters A-Z and a-z are considered alphabetic Default RESET will Revert User line status.

Minimum command length for RETRIEVE?

3

Command separator char.

;

Use fast renum if # lines > this value

999999

Default # columns for data shifts (Min 1)

4

Numer of columns/lines per Auto-Scroll

2

Notify tabs on external file change

All

Line Commands repeat limit (0=no limit)

0

Normal characters for screen display and for P.' picture literals

!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz

Display Char. for Invalid characters.

~



INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel

Done

Audible BEEP on Errors ?

If this box is checked, SPFLite will make a sound when an error message is displayed or if a key is pressed in a non-typing area. If this box is not checked, then no sound will occur. The specific sound produced is controlled by the Windows program event called Asterisk. To change the sound, go to Control Panel, Sounds and Audio Devices Properties, click on the Sounds tab, and click on the Windows/Asterisk program event. From the Sounds dropdown or the Browse button, select a sound you wish to use.

Visual BEEP on Errors ?

If Audible BEEP is turned off when the sound might be disturbing, it may still be helpful to have some indication when an error occurs. (This is especially true if a user were hearing-impaired and could not hear the Audible Beep.) If this option is chosen, SPFLite will flash the word **Command** in the upper left corner of the screen when an error occurs. It is allowable to have both options selected to receive both an audible and visual indication of an error.

Only 1 SPFLite running ?

If this box is checked, multiple executions (separate instances) of SPFLite will be prevented. If you attempt to open a file in a second SPFLite, it will detect the existing one running and cause your file to be opened as a new tab within the existing SPFLite execution. Once that is done, the second SPFLite will go away (actually, you will never even see it). Even when this option is set, a new SPFLite instance can **still** be opened by issuing an **OPEN** primary command from the existing SPFLite.

Re-Open last file(s) at Start ?

If this box is checked, and if SPFLite is started without a filename as a command-line operand, it will automatically reopen the last set of files that were being edited or browsed when SPFLite was last exited.

The prior set of open files is only remembered if SPFLite is exited using the Title Bar **X** button, or an **EXIT** or **=X** command. If all file tabs are closed one at a time with an **END** command or a right mouse click, SPFLite will restart without opening any prior files. Files must be actively open at the time of exit to be reopened later.

Delete to Recycle Bin ?

Files can be deleted by using the Delete line command **D** in File Manager, or by a **CANCEL DELETE** primary command. If this box is checked, deleted files are sent to the Recycle Bin. If this box is unchecked, deleted files will be "purged" (deleted without recourse) bypassing the Recycle Bin.

Even when the Recycle option is checked, recycling of files can be skipped and files permanently deleted with the Purge line command **U** and the **CANCEL PURGE** command, if desired.

Warn on modified View file ?

View allows you to modify the file as you like, but it prevents accidental saving of the modified data back to the file. However, sometimes it is easy to forget you are in View (because the two kinds of screens look so similar) and you might continue working, imagining that you were in Edit. If you then hit **END**, expecting the file to be auto-saved as Edit would do, you may be unpleasantly surprised to find all your changes are discarded. If this checkbox is enabled, an **END** command issued for a modified View tab will trigger a prompt to remind you that your changes will be lost. To avoid that, you can choose to terminate the **END** processing and return to View, where you can use the **CREATE** or **REPLACE** commands to save your data.

In addition, when you are in View mode and have modified the file, the Word 'View' in the left-hand status bar box will be displayed as **View** to remind you of the modified state.

Use DIR name as PROFILE when no File extension ?

If this is unchecked, then SPFLite will use the **DEFAULT** profile for files of this type.

If this option is checked, then the parent directory where the file resides will be used as the profile name. This can be useful when files are stored in directories that match a mainframe-like dataset naming convention.

See [What about files with no extension ?](#) in [Working with File Profiles](#) for more information.

Use WORD as the default for FIND/CHANGE commands ?

The **FIND** and **CHANGE** commands accept an optional "search context" parameter of **CHARS**, **WORD**, **PREFIX** or **SUFFIX**. In ISPF, if none of these keywords are present, **CHARS** is assumed by default. In SPFLite, the default can be either **CHARS** or **WORD**. To use **WORD** as the default search context, enable this checkbox. It is not presently possible to set a default of **PREFIX** or **SUFFIX**, since these choices are not as useful.

See [Finding and Changing Data](#) for more information.

Allow 2-D mouse selection without Shift/Ctrl/Alt ?

Experience has shown that it is easy (sometimes, **too** easy) to inadvertently select more than one line of text when you didn't intend to, unless you are very steady and careful with the mouse. If you mostly select text from single lines at a time, you may find it more productive to **not** check this box. In this mode, as a safeguard against doing this by mistake, you must hold down one of the Shift, Ctrl or Alt keys to perform multi-line selection.

However, if you often select multiple lines of text with the mouse, you may wish to check this box. It will allow text selection of more than one line of text (a "2D" text selection) without being required to hold down an extra key.

Only English letters A-Z and a-z are considered alphabetic

The WORD setting for a file's Profile allows you to customize what characters you wish to be considered as valid WORD characters. This check-box here can simplify the handling of ANSI standard international characters.

If this option is **disabled** (OFF), then when processing the WORD control string and **A-Z** is specified, all international uppercase characters will be automatically added; if **a-z** is specified, all lowercase international characters will be added. This is much simpler than having to individually type in all these characters, which are not defined in simple adjacent ANSI sequences.

Disabling this option will also cause the international characters to be added to the character string below, entitled Normal characters for P'.' Picture Literals, if that string has never been customized by you to include other special characters. See the discussion below for additional information. If you erase the "normal characters" list, then alter this check-box, the normal-characters list will get re-initialized to a value that agrees with how you have set the "only English" option.

If you don't use international characters in your data, but only English letters **A-Z** and **a-z**, you should **enable** this option. That will cause any international characters to be treated as "special" characters, rather than "alphabetic". It also means that **FIND** and **CHANGE** commands will find these characters with a picture code of **P' \$ '**, but they will **not** find them with picture codes of **P' @ '**, **P' < '** or **P' > '**.

In addition, the "only English" option will restrict commands such as **UC**, **LC**, **SC**, **TC** and **SORT** to only consider English letters **A-Z** and **a-z** when changing the case of characters or performing sorting and comparison operations on your data.

This implementation is consistent, so that both Picture handling, and the handling of commands like **UC** and **SORT**, either will or will not be restricted to English-only characters, based on how you set this option.

You may wish to enable this option if the presence of international characters might represent corrupted data rather than valid non-English letters. You may also wish to do this if you are using a font in which positions typically occupied by international characters contain other, non-alphabetic graphics.

When you alter this check-box, it takes effect immediately.

Default RESET will Revert User Line Status

When the **RESET** command is issued without any specific operands, it resets a default set of 'normal' items. If this option is selected, then the status of all User lines will be reset as part of that default set, similarly to how all Excluded lines are reset by default. If this option is left unchecked, the status of any User lines is left unchanged.

Minimum Command length for Retrieve ?

SPFLite saves command line entries for subsequent retrieval via the **RETRIEVE** command. The minimum-length value specifies the minimum length of a command before it will be saved for retrieval. This prevents filling the retrieve stack with unimportant entries which are just simply easier to re-enter. If you want commands of any length to be saved, set the minimum length value to 1.

Note that retrievable commands are saved between SPFLite sessions. A maximum of 50 commands can be

stored in the **RETRIEVE** stack.

Command Separator Char.

Multiple commands may be entered in a single SPFLite command line. This option specifies the delimiter character to be used to separate each command. The default command separator is the ; semicolon character, the same as in ISPF.

You cannot use a blank as the separator, nor attempt to delete the separator to make it a zero-length string, nor can you attempt to paste a hex value of X'00' into this field. You must define some character as the command separator, even if you don't plan to enter multiple commands on the command line.

Note that SPFLite's command separator works somewhat differently than in IBM ISPF

- IBM ISPF applies command separators unconditionally, regardless of what appears on the command line. So, if the separator is semicolon, and you had a quoted string with a semicolon in it, IBM ISPF will chop off the string right then and there, and then issue a command syntax error message. (A less complimentary way of saying this is, IBM applies its command separator **blindly**, and that's not always a good idea.)
- SPFLite honors quoted string values first, and *then* looks for command separators. So, if you leave the separator as ; semicolon you can still use semicolon in quoted strings without running into problems.

Most users will probably find the SPFLite approach works better for them than how IBM ISPF does it.

If you find the default command separator semicolon is an issue for you, you can set the separator to some seldom-used character. You would do the same if you didn't really want to define a command separator at all, but have to pick something because the rule above. You can use the ANSI popup in the KEYMAP to select some rarely used character that is convenient for you.

As a suggestion, the broken vertical bar ; character is a perfectly good ANSI character (hex value A6) but is not a "standard ASCII" value (one that is less than hex 7F). You could map this character to (say) the **Ctrl backslash** key, and that would make a nice separator character that you're not likely to see or use in your own data. The mapping string for this would look like [;] in [KEYMAP](#). You can use the **Show Character Map** button to get the ANSI popup; just click on the ; character and then paste it into the mapping string (normally by using the **Ctrl-V** key, unless you have mapped the [\(Paste\)](#) function to something else). Be sure to add the [and] before and after the ; character, of course.

Bear in mind that the Options screens are [Windows](#) dialog GUI displays, and if you have mapped some special character using KEYMAP, you can't use it in a Windows dialog, but only within SPFLite itself. If you have a special keyboard layout that Windows knows about, either a foreign layout, or a custom one created by the Microsoft Keyboard Layout Creator or KbdEdit, you can directly use those keys anywhere, and do not need to map them in SPFLite unless you wish to override them for SPFLite-specific functions. (An Internet search will find more information regarding the Microsoft Keyboard Layout Creator and KbdEdit.)

So, if you have some non-ASCII special character (higher than X'7F') you'd like to use, you can always enter your mapped key on an SPFLite command or data line, as described above. But, to actually set the Command Separator, you have to do it in a way that Windows understands. The easiest way to do this is to bring up the SPFLite ANSI display, and left-click on the character you want to use. Then, in the Options GUI where you need to set this, go to the Command Separator Char field, delete the current contents, and press **Ctrl-V** to enter the special character. Note that when you are in a [Windows](#) dialog or GUI, [Windows](#) controls how the keyboard is interpreted, and so within such a dialog or GUI, **Ctrl-V** **always** means Paste, no matter what you mapped this key to in SPFLite.

Use fast renum if # lines > this value

When an edit session is loaded with a large file, it can create a noticeable delay when inserting/deleting lines or doing Exclude/UnExclude operations. This is because SPFLite has always 'numbered' the lines with their actual

physical location within the file. This means constant renumbering is required to adjust these numbers. On large files, this can create noticeable delays.

SPFLite can now alternatively use a method like the original ISPF where lines are not numbered consecutively and during Insert/Delete operations only lines within the 'neighborhood' of the affected lines are renumbered to accommodate the change.

All functions using line numbers are unaffected, however some users are not comfortable with these 'odd' line numbers. Everything is still in strict numerical order, just not numbered with a single incremental value.

This Options setting allows **you** to specify a value for what will be considered a **large** file, and thus which files will be numbered using this new 'fast' renum option.

If you are not bothered by this new line numbering, set this value to **0** and all files will use the new numbering. Or, set it to 999999 and all files will continue to use the old renumbering (assuming you have no 1,000,000+ line files). Or set it to any value in between based on your own delay experiences on **your** system.

Default # columns for Data Shift (Min 1)

This setting specifies the default number of columns to be shifted when using one of the line shift commands {[\(U\)](#), [\(<\)](#), [\(>\)](#), or [\(I\)](#)} and no specific number of columns is specified on the command itself. IBM ISPF does not allow this default to be set by the user, but enforces a fixed default of 2 columns. In SPFLite, you can make the default number of columns any positive value you wish.

Number of Columns/Lines per Auto-Scroll

The value entered here will be used to control the number of lines (for Vertical scrolling), or characters (for horizontal scrolling) to be performed for each 'click' of the mouse wheel. If **0** (zero) is entered, then no action is taken when the Mouse Wheel is moved.

Note: The value set here is also used as the scroll amount by the keyboard primitives [\(ScrollLeft\)](#), [\(ScrollRight\)](#), [\(ScrollUp\)](#) and [\(ScrollDown\)](#). In case you wish to disable Mouse Wheel scrolling by setting this number to 0, the keyboard scrolling will be done with a default amount of one line or one column.

Notify tabs on external file change

SPFLite can be directed to inform you when a file you are working on in an Edit or Browse session has been modified by some process outside of SPFLite itself. You have control over what conditions under which you will receive a notification.

The permanent file notification level is defined here in the General tab of the Global Options dialog, and is either **ALL**, **NONE** or **EDIT**.

The [NOTIFY](#) command is used to set the temporary notification level for files opened in SPFLite that are modified by an external process. The setting you choose on **NOTIFY** only lasts until the next **NOTIFY** command, or until SPFLite is terminated.

See [Handling External File Changes](#) for more information.

Line Commands repeat limit

Occasionally, as you enter line commands, you may inadvertently end up with a line command entered which still has some remaining digits of the line number itself. This could result in a wildly incorrect 'repeat factor'. To prevent this, you may specify a number here which will 'cap' the repeat factor. Simply enter here the largest number you would probably ever enter as a repeat factor. Any command entered which exceeds this value will be rejected.

Normal characters Screen display and for P'.' Picture Literals

The characters in this text box define what characters are "displayable" characters in your currently chosen screen font. The characters are also used by the Picture character ' . ' (dot) when performing Picture searches.

When SPFLite is displaying your text on the screen and a character is **NOT** in this list, SPFLite will substitute the value you give in "[Display Char. for invalid characters](#)" below.

Note that the dot Picture is used to locate characters which are considered "non-displayable" on the screen. That is, a Find Picture of ' . ' dot will find characters that are **not** in the list of 'Normal' characters you see here. Be careful not to get this confused.

The default list contains the basic ASCII character set, and if you have chosen the **Include International characters for WORD/Alphabetic Pictures** option, the normal ASCII international characters.

You can alter this as needed to incorporate additional characters based on your local requirements, so they will not be considered "non-displayable". Simply add or remove characters as needed.

Note: If you alter the **Include International characters for WORD/Alphabetic Pictures** option, this field will automatically be adjusted for the presence/absence of those characters if you have not customized the field previously. If you have customized the field, other than via the previous option, then this field is not altered so as to not lose your customization.

Note: If you completely erase the Normal Characters field and close the Options dialog, SPFLite will repopulate the Normal Characters string with all valid displayable characters. When it does this, if the International Characters check-box is enabled as of that time, the repopulated string will contain international letters; if the check-box is not enabled, the repopulated string will not contain international letters.

Displayable vs. non-displayable characters were more of an issue for the 3270-based mainframe ISPF editor, because characters that really are non-displayable on a 3270 can lock up the screen. SPFLite doesn't have this problem. Very few hex values are non-displayable on a PC, depending on the font used, and no character value will lock up the PC screen. However, with some fonts, non-displayable characters can cause characters to be mispositioned on the screen. If you use one of the supplied RASTER fonts, almost every hex value is displayable.

In terms of what you might do with a Find Picture of p' . ' in SPFLite, it's more likely that what you are looking for is, not "non-displayable" characters, but "unusual" ones. What makes a character "unusual" ? That's for **you** to decide, which is why you can configure it here. Just remember that the "non-displayable" or "unusual" characters are the ones **not** in this list.

Display Char. for invalid characters

When SPFLite displays characters on the screen, and a character is encountered which is **NOT** in the list of [normal screen characters](#), this option allows you to choose what character to be used as a substitute. The original IBM ISPF used a Period (.) and that is the default for this value. Feel free to make it any other infrequently used character of your choice. The choice **must** be a character from the [normal screen characters](#),

Cancel and Done Buttons

One of the CANCEL or DONE buttons at the bottom of the dialog box should be selected when you have completed your changes. The **Cancel** button will discard all changes you have made and exit the Options process. The **Done** button will save your changes and return to the Editor.

Options - File Manager

Index of File Manager Options

[Close SPFLite with last tab ?](#)
[Confirm file Deletes ?](#)
[Display File Manager Help ?](#)
[Highlight Recent / Active dates?](#)
[Specify optional columns and their order](#)
[Enter filename extensions considered to be Non-Text files](#)
[No. files in recent lists](#)
[Default File Open Directory](#)
[Width of Command Line area \(Min 5\)](#)
[Width of Note column area \(Min 5\)](#)
[Initial FM column sort](#)
[Directory Placement](#)

General **FM** Submit Screen KBD Status Schemes HiLites

Close SPFLite with last file tab?

Confirm file Deletes?

Display File Manager Help?

Highlight Recent / Active dates?

Enter the Optional desired columns and their order

DATETIME,SIZELONG

Valid column names to choose from are:

EXT, DATE, DATETIME, SIZESHORT, SIZELONG, LINES and NOTE

Enter filename extensions considered to be Non-Text files

exe, dll

Number of entries in recent lists

35

Default File Open Directory

Last Used Dir

Width of Line Command area (5 to 20)

6

Width of Note column area (5 to 20)

20



INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel

Done

Close SPFLite with last tab ?

If this box is checked, then an [END](#) command on the last open Edit/Browse tab will terminate SPFLite..

If not selected, then following the close of the last active Edit/Browse tab, you will be switched to the File Manager Tab. Note also that if you right-click any file tab or File Manager tab, it is treated exactly the same as if an **END** command were issued for that file or for the File Manager.

Confirm file Deletes ?

If this box is checked, then each Delete or Purge action selected for a file will result in a confirmation prompt before the file is actually processed. Files are deleted with the **D** line command and the **CANCEL DELETE** primary command; and are purged with the **U** line command and the **CANCEL PURGE** primary command.

The confirmation will tell you whether the file will be Recycled or will be Purged (permanently deleted) depending on the Recycle option you have set, and gives you a chance to change your mind by clicking **No**.

If not selected, the Delete or Purge will proceed without this prompt. Because this helps you avoid deleting files by mistake, you are encouraged to enable this option.

Display File Manager Help ?

If this box is checked, when the File Manager is activated it will display a Help legend at the bottom of the screen, showing File Manager line command codes and other information. Once you have used File Manager for a while and can remember these codes, the Help display lines can be removed to reclaim some screen space for displaying more file names, by unchecking this box.

Highlight Recent / Active dates ?

If this box is checked, then the file dates in the FM list will be color coded to highlight the files which have been accessed most recently. This highlighting is based on two date 'boundaries':

The Active Date/Time

This is the Date / Time the current SPFLite session was started. Files which have been modified **since** this time will have their Date / Time value highlighted using your specified color values for RED highlighting.

The Recent Date/Time

This is set to 12:01 AM of the previous day (based on when SPFLite is started). Files which have been modified **since** this time will have their Date / Time value highlighted using your specified color values for BLUE highlighting.

For information on setting the specific colors for RED and BLUE highlighting see [Options -> Screen -> Screen Colors](#)

Specify optional columns and their order

In this box, enter from left to right, the optional File Manager information columns you wish to be displayed. The available columns are:

- **EXT**

This column will contain the file extension as a separate column. Only the first 8 characters of an extension will be displayed if the extension is longer. The extension will still be displayed normally as part of the normal filename display.

- **LINES**

This column will contain the number of data lines in the file. This will only occur if:

- The Profile for the file type is set to **STATE ON**.
- The file actually **has** a valid STATE file created.

Details on STATE handling can be found in [Saving the Edit STATE](#)

- **DATE**

This column will have the last modified date of the file in the format. e.g. YYYY-MM-DD .

- **DATETIME**

This column will have the last modified date of the file in the format. e.g. YYYY-MM-DD HH:MM .

- **SIZESHORT**

This column will have the size of the file in abbreviated Byte format. e.g. 789, 12.3K, 123.4M etc.

- **SIZELONG**

This column will have the size of the file in detailed Byte format. e.g. 9,999,999,999.

- **NOTE**

This column is available for you to record small notes, memos, etc. related to the file. It will be a scrollable window, the maximum size of the note is 255 characters.

The selected columns will appear to the right of the filename column, in the order you specify them. The filename

column width will be adjusted to use the remaining width of the screen after allocating your requested columns.

Enter filename extensions considered to be Non-Text files

When using the FF (Find in Files) command in File Manager, there are certain file types you probably do not wish to search, such EXE and DLL files, and other binary file types. If you specify their file extensions here, the FF command will not search files of these types; that will speed up the search process, and will prevent SPFLite from trying to create default profiles for file types you never edit.

Because file types you do not want searched by FF are most likely binary files, they are the same kinds of files that you would not want to be modified by a NUMBER or RENUM sequence numbering command. For that reason, the sequence numbering commands consider any non-text file type you list here as being "exempt" from being numbered or renumbered.

See [Working with Sequence Numbering](#) for more information.

No. files in recent lists

This setting specifies the maximum number of files to be retained in the **Recent Files** and **Recent Paths** file lists. The lists will automatically retain this number of entries, and remove older, less used entries from the list. The maximum value you can use is 99.

Default File Open Directory

This setting allows you to choose one of two Directories to use as the Default when SPFLite attempts to Open or Save a file.

- **Working Dir** - SPFLite will use the current working directory at the time it is started. If you start SPFLite from the command line, that is what the working directory is. If you start SPFLite from an icon on the Windows desktop, there is a Start Directory you can set in the icon's Properties window; the Start Directory is the SPFLite Working Dir.
- **Last Used Dir** - SPFLite will use the last directory where it Opened or Saved a file.

Width of Command Line area (Min 5)

If you often use longer FM line commands, you can widen this field, which defaults to a minimum length of 5. Regardless of the width set for this field, commands you enter will scroll as you type them, so you can still enter any command you need.

Note: The width of the File Manager's line command area (set in File Manager Options) and the width of the Edit screen's line command area (set in Screen Options) are different settings. Also, the File Manager's line command area will scroll horizontally, while the Edit screen's line command area does not scroll.

Width of Note Column area (Min 5)

If you utilize the Note support of File Manager, this setting allows you to specify how much space on the file manager screen should be allocated to the display of the Note field. This does not restrict the size of the note itself, the Note field will scroll to allow viewing/editing of the data.

Options - Submit

SPFLite Global Options

General | FM | **Submit** | Screen | KBD | Status | Schemes | HiLites

Prototype command line to be used by SUBMIT
echo "~-i"

Working Directory to be used by SUBMIT
D:\Documents\SPFLite\Jobs

CMD.EXE flags used by SPFLite CMD command
/K

CMD.EXE flags used by SPFLite RUN command
/K

Trigger key for the SUBMIT Include statement
#INCLUDE

Column number where above Include Key must appear
1

 INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel | Done

Prototype Command Line to be used by SUBMIT

The [SUBMIT](#) command will extract the data lines to be submitted, create a temporary file containing them, and then will invoke your desired command processor to process this file. This text box is where you specify exactly what the prototype for this command should look like. The command string will be issued as it appears here, after variable substitution has been completed. See the [SUBMIT](#) command and the "[Working with the SUBMIT Command](#)" for full details on using this facility. To facilitate the submission of jobs to the Hercules emulator, you may use the SPFSUBMIT.EXE program supplied with SPFLite.

Working Directory to be used by SUBMIT

During **SUBMIT** processing, temporary files used during the process will be stored in this directory. SPFLite will

automatically clean up files in this directory once they are over two days old. In this way, they will remain available for any use you might have for normal purposes, but will not accumulate after they have outlived their usefulness.

CMD.EXE flags used by SPFLite CMD command

The **CMD** command opens a Windows command window to process the specified command. This option allows you to control the specification of any optional operands for CMD.EXE itself for this function. Although any valid CMD.EXE options may be specified, the primary use of this option will be to specify **/c** (to close the window following execution) or **/k** (to keep the window open following execution).

You may find the **/c** flag more useful than **/k** in most cases. If you use the **/k** option, and you wish to EXIT from the command prompt window, you may need to issue the EXIT command twice.

Note: For a list of available CMD.EXE flags, bring up a Windows command prompt, enter the command **CMD /?** and press Enter.

CMD.EXE flags used by SPFLite RUN command

The **RUN** command opens a Windows command window in which to execute the script. This option allows you to control the specification of any optional operands for CMD.EXE itself for this function. Although any valid CMD.EXE options may be specified, the primary use of this option will be to specify **/c** (to close the window following execution) or **/k** (to keep the window open following execution).

Trigger key for the SUBMIT Include statement

This specifies the trigger string which is used to identify the unique INCLUDE statement. When present in the job-stream being submitted, the INCLUDE statement specifies the name of a local file to be inserted into the job-stream to replace the actual INCLUDE statement. You may specify any unique string you prefer. The default for this value is **#INCLUDE**. The trigger value will be looked for in the column number specified by the next option. (Default 1)

The format for the INCLUDE statement is just the Trigger Key followed by the filename to be included. The name can be a simple filename, in which case it must be located in the same folder as the current edit file, or it may be a fully qualified path/filename. The name may optionally be enclosed in quotes (single or double). Examples:

```
#INCLUDE MYFILE.TXT
#INCLUDE "C:\Users\Documents\Files\AnotherFile.txt".
```

Column number where the above Include key must appear.

This specifies the column number where the above trigger value must appear. The occurrence of the trigger value anywhere else in the line will **not** activate INCLUDE processing. The default value for this setting is **1**.

Options - Screen

Index of Screen Options

[Cursor handling and Blink](#)
[Vertical Cursor in Insert Mode](#)
[% Height Normal Cursor and Insert Cursor](#)
[Font Name, Font Pitch and Choose](#)
[Column Marker Line Color](#)
[# Keyboard help lines to show](#)
[Cross-Hair Cursor Rulers](#)
[Alternating Background Colors](#)
[Hi-Lite Cmd line keywords?](#)
[Width of Line numbers \(5-8\)](#)
[Screen Colors - Click button to change](#)
[Indicating file-modified status in the tab header](#)

General	FM	Submit	Screen	KBD	Status	Schemes	HiLites																																								
<input checked="" type="checkbox"/> Vertical cursor in Ins. Mode. <input type="checkbox"/> 24 % Height Normal Cursor. <input type="checkbox"/> 100 % Height Insert Cursor. <input type="checkbox"/> DejaVu Sans Mono Bold <input type="checkbox"/> 12 <input type="checkbox"/> Choose <input type="checkbox"/> Column Marker Line Color				<input type="checkbox"/> Horizontal cursor ruler? <input type="checkbox"/> Vertical cursor ruler? <input checked="" type="checkbox"/> Alternating Background screen colors? <input checked="" type="checkbox"/> Hilight Command line keywords? <input type="checkbox"/> 0 # K-Board help lines to show? <input type="checkbox"/> 8 Width of Line Numbers (5-8)																																											
FG BG1 BG2																																															
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Line Numbers High Intensity</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/></td> </tr> <tr> <td>Line Numbers Low Intensity</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/></td> </tr> <tr> <td>Active Tab Modified</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/></td> </tr> <tr> <td>Active Tab Unmodified</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/></td> </tr> <tr> <td>Inactive Tab Modified</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> </tr> <tr> <td>Inactive Tab Unmodified</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> </tr> <tr> <td>PFK Help Lines</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/></td> </tr> <tr> <td>Status Line</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> </tr> <tr> <td>FM Tool Bar</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/></td> </tr> <tr> <td>Error Messages</td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/></td> <td><input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/></td> </tr> </table>								Line Numbers High Intensity	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>	Line Numbers Low Intensity	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>	Active Tab Modified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	Active Tab Unmodified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	Inactive Tab Modified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	Inactive Tab Unmodified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	PFK Help Lines	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/>	Status Line	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	FM Tool Bar	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/>	Error Messages	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>
Line Numbers High Intensity	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>																																												
Line Numbers Low Intensity	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>																																												
Active Tab Modified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>																																												
Active Tab Unmodified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: yellow;" type="color"/>																																												
Inactive Tab Modified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>																																												
Inactive Tab Unmodified	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: purple;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>																																												
PFK Help Lines	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: gray;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightblue;" type="color"/>																																												
Status Line	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: black;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>																																												
FM Tool Bar	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: darkblue;" type="color"/>																																												
Error Messages	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: red;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: white;" type="color"/>	<input style="width: 20px; height: 20px; border: 1px solid black; background-color: lightgreen;" type="color"/>																																												



INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel

Done

Cursor handling and Blink

SPFLite uses the standard Windows cursor support. To control the Blink rate (or turn it right off) go to the system's Control Panel => Keyboard settings panel. Note that changing this setting affects the cursor blink rate for your entire system, not just SPFLite.

Vertical Cursor in Insert Mode

If checked, then a vertical bar cursor will be used when the keyboard is in Insert Mode rather than a square cursor (which can be adjusted by the next two items in this dialog). If this option is checked, the value chosen for **% Height Normal Cursor and Insert Cursor** will be ignored when in Insert mode.

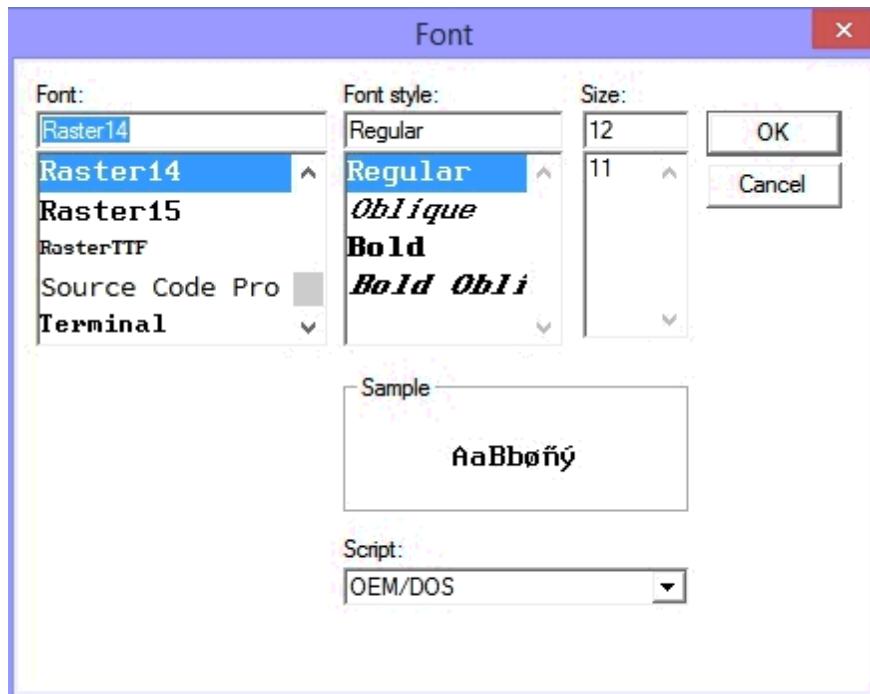
% Height Normal Cursor and Insert Cursor

These two values control the appearance of the screen cursor in Normal and Insert mode. The value given is a % of a full height Blob cursor (i.e. one which fills the character space). The value can range from 20 to 100%. Note: depending on the screen font you choose, values below 20% tend to be either invisible, or fragmented dots. So, if you have an underscore-like cursor and find that your cursor disappeared, you may need to adjust

these settings. Note the setting for the Insert cursor is ignored if you have chosen to use a Vertical cursor in insert mode (above)

Font Name, Font Pitch and Choose

These three related items allow you to choose what screen font will be used for the display. You can either directly specify a specific name and pitch (e.g. TERMINAL / 11) or select the **Choose** button to invoke a standard Windows Font Selection Dialog. Note that **only fixed-width (non-proportional)** fonts are supported.



SPFLite provides a number of fixed fonts to use, if you need something besides the system-provided fonts.

You will notice many examples in this Help document use the RASTER font, available as an SPFLite download. There are three Raster fonts available. Raster is 15 points high. Raster14 is 14 points high, and allows a few more lines on a screen. RasterTTF is the Raster font converted to TrueType format. RasterTTF has the same character set as Raster, but is intended primarily for printing files. RasterTTF has the advantage of being scalable, while Raster and Raster14 are crisper fonts for the screen. Later versions of SPFLite have added yet smaller versions of Raster, namely Raster13, Raster12 and Raster11. Check the SPFLite download page for the most current down-loadable fonts.

Column Marker Line Color

This color select box is used to choose the color to be used for column markers. Column markers are faint vertical lines visible on the screen at specified columns to provide column alignment 'clues'. They are specified by the [MARK](#) line command.. See that command for more info. Simply click on this box to go to the color selection dialog to make your choice.

Horizontal Cursor Ruler?

Vertical Cursor Ruler?

A hardware option on the original 3270 terminals, and some 3270 emulators, provides 'cross-hair' cursors. These are visible full height/width lines which followed the cursor location making the position very easy to track. It's a personal option; some users love them and other find them distracting. SPFLite supports this style and allows you to optionally select either or both of the vertical and horizontal lines. The lines are drawn in the MARK line color, which is set in the left-hand column. A sample of cursor rulers is shown below.

testdata.txt - SPFLite(v8.0.4126) - D:\Documents\Test Data\testdata.txt

File Manager testdata.txt Command > Scroll > CSR

```
*****
000001 1111111111 1111111111 "Fred"
000002 2222222211 2222222222 fred
000003 3333333333 3333333333 fred 3      ' two_
000004 4444444444 4444444444 fred 4      ' one_two_three
000005 5555555555 5555555555 fred 5      ' nuther comment
000006 6666666666 6666666666 fred 7      fred
000007 7777777777 7777777777 fred 8
000008 8888888888 8888888888 fred 9
000009 9999999999 9999999999 FRED 10
000010 0000000000 0000000000 FRED 11=1
```

Edit * L 000005 C 12 Lines: 245 Cols 1 to 73 Bnds: MAX OVR T CS S- Line Len 0067

Alternating Background Colors

If selected, SPFLite will use alternating bands (3 lines each) of background colors when displaying the screen. This is similar to the use of 'green banded' paper for printouts in the past. It assists in maintaining eye positioning when scanning text lines. If you activate this option, the choice of the alternate color is made in the Screen Colors section below. A sample of a screen with alternating background colors is shown below.

NOTE: Hercules users are familiar with the Hercules Remote Print Spooler utility program **HercPrt**, which (among other things) has the ability to take a SYSOUT file and format it into a PDF file that looks remarkably like a computer printout on "green bar" paper - complete with sprocket holes and perforation! This is cute and very clever, but these PDF files use a large amount of disk space. By using alternating background colors (along with EOLAUTO and PAGE ON mode), it is possible to very closely simulate the effect of a HercPrt-formatted file without the PDF disk-space overhead. You will still be editing or browsing ordinary text files in native mode, but the display will have the look and feel of paging through an actual hard copy printout.

If you wish to match the same colors generated by the HercPrt utility, make the main background color (BG1) white, and the alternative background (BG2) color a light green.

testdata.txt - SPFLite(v8.0.4126) - D:\Documents\Test Data\testdata.txt

File Manager testdata.txt Command > Scroll > CSR

```
*****
000001 1111111111 1111111111 "Fred"
000002 2222222211 2222222222 fred
000003 3333333333 3333333333 fred 3      ' two_
000004 4444444444 4444444444 fred 4      ' one_two_three
000005 5555555555 5555555555 fred 5      ' nuther comment
000006 6666666666 6666666666 fred 7      fred
000007 7777777777 7777777777 fred 8
000008 8888888888 8888888888 fred 9
000009 9999999999 9999999999 FRED 10
000010 0000000000 0000000000 FRED 11=1
```

Edit * 2014-04-29 13:23:28 Lines: 245 Cols 1 to 73 Bnds: MAX OVR T CS S- Line Len 0067

Hi-Lite Cmd line keywords?

With the large number of keywords supported by some commands, it sometimes becomes difficult to remember them all and the need to sometimes enclose them in quotes to use as literals. This can cause some confusing error messages to appear. If this item is selected, then recognized keywords on the command line will be displayed in high-intensity, non-keywords in lo-intensity. For example"

```
*****
***** Top of Data *****
000001 1111111111 1111111111 "Fred
000002 2222222211 2222222222 fred
000003 3333333333 3333333333 fred 3
000004 4444444444 4444444444 fred 4
000005 5555555555 5555555555 fred 5
000006 6666666666 6666666666 fred 7
000007 7777777777 7777777777 fred 8
' two_
' one_two_three
' nuther comment
fred
```

The keywords 'change', 'prefix', and 'all' are hi-lighted, the remaining operands are left in lo-intensity.

K-Board help lines to show

You can choose to have a summary of the local commands keys displayed at the bottom of the screen. You can choose from 0 to 5 lines for the display, where 0 means to omit the display altogether. Only keys which have been defined with a Key Label will be displayed. See "[Keyboard Customization](#)" for how to set up Key Labels. The keys are displayed in logical key order rather than strictly by their alphabetic sorting.

e.g. the keys S-F1, F1, F10, A-F10, F2 and S-F2 would be displayed in order F1, S-F1, F2, S-F2, F10, A-F10.

Width of Line numbers (5-8)

In contrast to ISPF, which has a fixed Edit sequence number width of 6, you can make the size of this area longer to accommodate files with large numbers of lines, or shorter to allow more text data to be visible when working on displays with screen-size constraints.. You may set a value here from 5 to 8 characters.

If you change the width of this field while you have files open that are displaying "special" lines, like Profile values, MARK, MASK, BOUNDS, etc. some of these special lines "markers" may not be displayed correctly. To correct this, issue a RESET command and redisplay the lines in question. Typically, you will set this value to what you prefer, and then seldom if ever change it again.

Note: Some sequence area "markers" will not fit with their normal appearance when the width is set to 5. When this happens, SPFLite will shorten the marker to display it as best it can. For example, **=PROF>** becomes **PROF>**. For most markers, it will be obvious what is meant. For **NOTE** and **xNOTE** lines, the markers had to be reformatted. In 5-column mode, these will now appear as **=##=>** and **=x##=>** respectively.

Note: The width of the File Manager's line command area (set in File Manager Options) and the width of the Edit screen's line command area (set in Screen Options) are different settings. Also, the File Manager's line command area will scroll horizontally, while the Edit screen's line command area does not scroll.

Note: A few of our more ambitious users have attempted to edit very large files, approaching and in a few cases exceeding one million lines. You are certainly welcome to try this. However, editing files that large is what most people use databases for, not text editors - and such usage pushes the limits of what SPFLite is capable of doing. If you intend to do that, and you are reading this section here because you want larger sequence numbers to accommodate your file, we have a few recommendations that may be of benefit:

- Be certain you have a current backup of your file before opening it with SPFLite.
- Editing extremely large files means that virtual storage will be a constraint. While your edit is in progress, you should close all other Windows programs that are not absolutely necessary.
- Limit the number of edit sessions within SPFLite that are running at the same time you are editing the large file, preferably having the large file open as the only edit session.
- Be sure to close your edit session before exiting SPFLite, or else be sure the General Options check-box

that says "Reopen last files" is left unchecked. Otherwise, the next time you start SPFLite (perhaps for some other purpose) it will reopen your large file, which may be quite time-consuming.

- Editing extremely large files will have a noticeable performance impact. Some operations will take longer than you might expect. Be prepared to be patient. You may also see the "loop detected" message because of the time required to complete some operations. In most cases, you can simply click OK to proceed.

Screen Colors - Click button to change

The bottom of this tab is used to select the colors you wish to use for the screen display. These colors are used for displaying the non-data portions of the screen, the color settings for the data itself are specified on the Options -> Schemes tab.

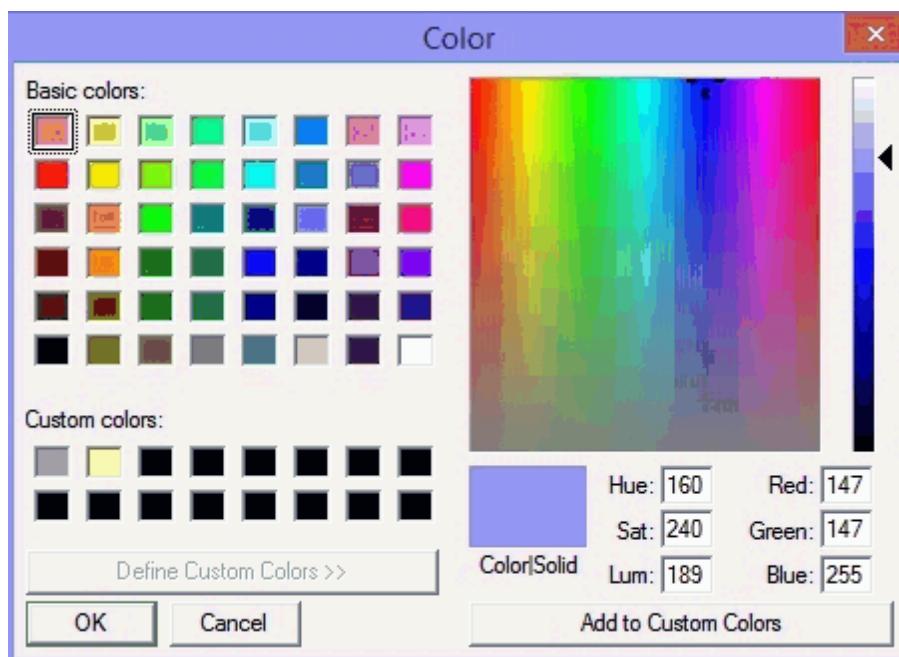
Each of the 10 entries has an FG, BG1 and BG2 color selection. The FG specifies the color of the foreground text itself, the BG1 and BG2 specify the background color(s) Only BG1 is used if Banding is Off, both BG1 and BG2 are used if banding is On.

Note: Some of these entries may never actually use the BG2 entry. When color selections are made, the name of each entry will be displayed in your selected colors so that you receive immediate feedback on what your choices will look like.

To alter a color, click the displayed color box and you will be shown the standard system color chooser dialog.

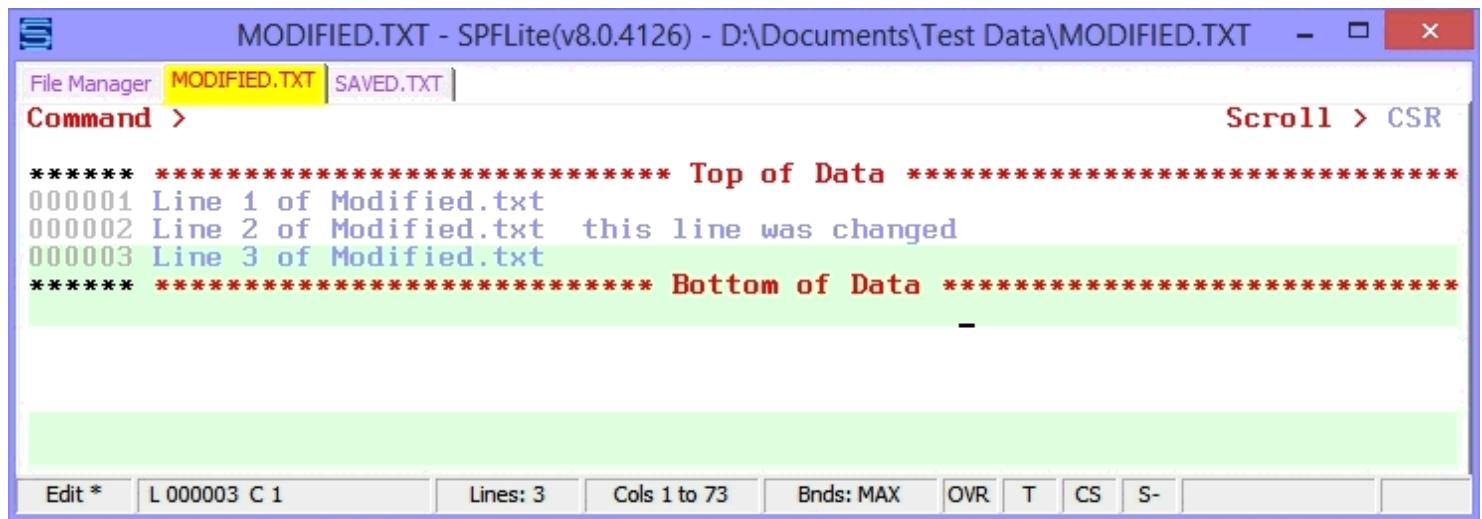
Custom Colors

Note: When you create custom colors using the color chooser, SPFLite will "remember" these custom colors for your future use. This means you can create and save 16 custom colors.



Indicating file-modified status in the tab header

Here is an example of how these color choices would appear in the main edit window. The current file tab shows red lettering because **MODIFIED.TXT** is modified, while **SAVED.TXT** is in blue lettering because it is unmodified.



Options - Keyboard

SPFLite Global Options

General | FM | Submit | Screen | **KBD** | Status | Schemes | HiLites

Scroll left/right with Arrow Keys?

Scroll up/down with Arrow Keys?

Keyboard starts in INSERT mode?

Reset INSERT mode on Attention?

 INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel

Done

Scroll left/right with Arrow keys

Scroll up/down with Arrow keys

If selected for the appropriate direction (left/right or up/down), then when the cursor is moved to the edge of the actual text area, further movement will cause the screen to scroll data into view based on the cursor direction.

If not selected, then the cursor will wrap to the opposite edge of the screen as it does on 3270 terminals.

KB Starts in INSERT mode?

If this box is checked, then the keyboard will be in INSERT mode as the default when SPFLite is started.

Editing in Insert mode is typical for Windows-based editors, while editing in Overwrite mode is how the mainframe 3270-based ISPF operates. Even with the many word-processing enhancements available in SPFLite, most users will find it more practical to operate SPFLite in Overwrite mode most of the time rather than in Insert mode.

Reset INSERT mode on Attn?

If selected, then when any "attention" key is pressed, then Insert mode will be reset to the value specified in the previous option. If not selected, the insert mode will not be altered.

An "attention" is recognized when either an [\(Enter\)](#) key is pressed or a primary edit command is issued, whether that primary command is typed in on the primary command line or is executed from a mapped key.

IBM ISPF normally clears an active insert mode when you press Enter or a PF key. SPFLite does not have the equivalent of a 3270 ATTN key.

Options - Status

SPFLite Global Options

[General](#) | [FM](#) | [Submit](#) | [Screen](#) | [KBD](#) | **Status** | [Schemes](#) | [HiLites](#)

You may customize the contents of the Status Bar here.

Simply choose below which fields you want to appear.

Enter your desired Status Bar fields from Left to Right

MODE,LINNO,LINES,COLS,BNDS,INSOVR,CHANGE,MISC,SELECT,SOURCE,EOL

Valid field names to choose from are:

BNDS, CAPS, CASE, CHANGE, COLS, EOL, INSOVR, LINES, LINNO, MISC, MODE, SELECT, SOURCE and STATE

Refer to Help => Working with SPFLite => Status Bar Contents for more info.



Status Bar Configuration

This tab allows you to specify **what** status bar boxes you wish to appear on the bottom of the screen, and their relative position - left to right.

Simply enter the names of the boxes you desire separated by commas. The valid box names are:

BNDS, CAPS, CASE, CHANGE, COLS, EOL, INSOVR, LINES, LINNO, MISC, MODE, SELECT, SOURCE and STATE.

Details of the contents of these boxes may be found in "[Status Bar Contents](#)".

Options - Schemes

SPFLite Global Options

General | FM | Submit | Screen | KBD | Status | **Schemes** | HiLites

Scheme FG BG1 BG2

AUTO Colorization Scheme 1	
AUTO Colorization Scheme 2	
AUTO Colorization Scheme 3	
AUTO Colorization Scheme 4	
AUTO Colorization Scheme 5	
AUTO Colorization Scheme 6	
AUTO Colorization Scheme 7	
AUTO Colorization Scheme 8	
AUTO Colorization Scheme 9	
AUTO Colorization Scheme 10	
AUTO Colorization Scheme 11	
AUTO Colorization Scheme 12	
AUTO Colorization Scheme 13	
AUTO Colorization Scheme 14	
Normal Text Lo Intensity	
Normal Text Hi Intensity	

GPL Free Software

INI File is:

Test Color Scheme Setting

This tab allows you to set the default color schemes to be used to display the actual Edit text data. If you are not using the automatic [colorization option](#), then only the first two items are of interest. (Normal Text Lo Intensity and Normal Text Hi Intensity)

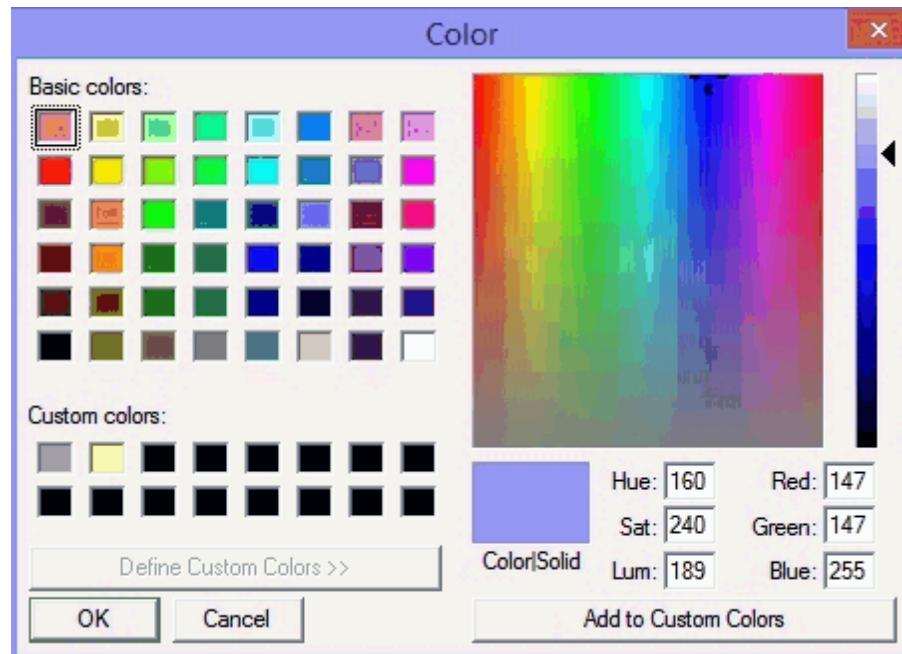
If colorization **is** active, then it is [here](#) where you specify the various color schemes referenced by the [AUTO](#) files.

Each of the entries has an FG, BG1 and BG2 color selection. The FG specifies the color of the foreground text itself, the BG1 and BG2 specify the background color(s) Only BG1 is used if Banding is Off, both BG1 and BG2 are used if banding is On.

To alter a color, click the displayed color box and you will be shown the standard system color chooser dialog.

Custom Colors

Note: When you create custom colors using the color chooser, SPFLite will "remember" these custom colors for your future use. This means you can create and save 16 custom colors.

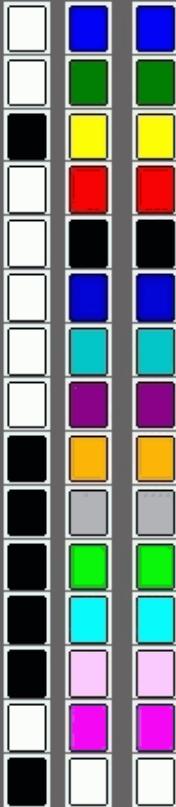


Created with the Personal Edition of HelpNDoc: [Free PDF documentation generator](#)

Options - Hi-Lites

HiLite Color Name FG BG1 BG2

(B) BLUE
(G) GREEN
(Y) YELLOW
(R) RED
(K) BLACK
(N) NAVY
(T) TEAL
(V) VIOLET
(O) ORANGE
(A) GRAY
(L) LIME
(C) CYAN
(P) PINK
(M) MAGENTA
(W) WHITE



The letters in parens are used as color-setting codes in SPFLite macros.



Specifying Hi-Lite Color Selections

SPFLite allows you to Hi-Lite sections of a file, much like what you would do with a Yellow hi-lighter on real paper. For more information on this feature see ["Working with Virtual Hilighting pens"](#)

This tab allows you to set the colors to be used for this Hi-Liting function.

The names of the colors cannot be altered, however you are free to make the displayed color whatever you please. If you'd like RED to be a lovely shade of Brown, go right ahead.

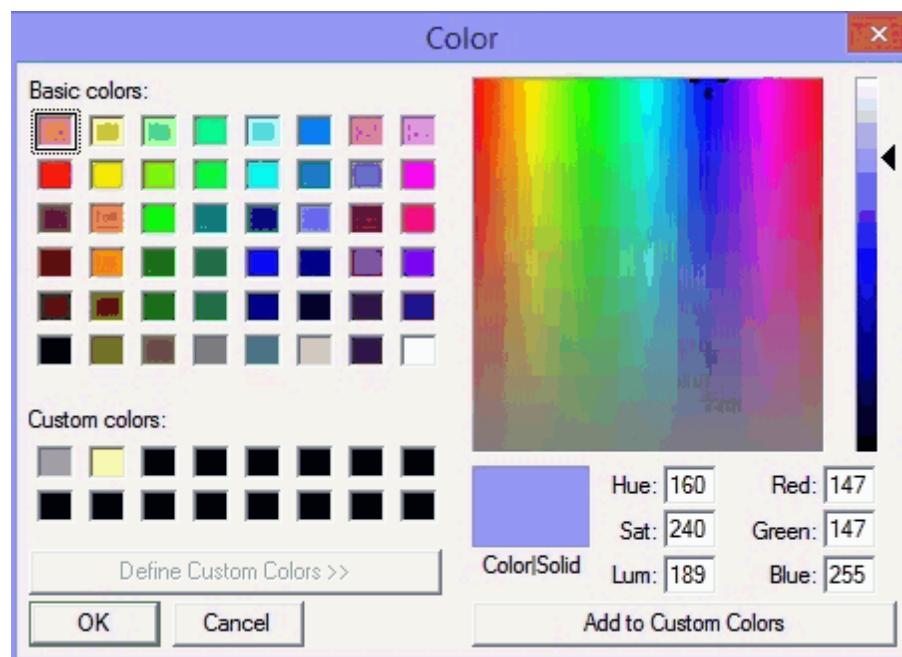
Each of the entries has an FG, BG1 and BG2 color selection. The FG specifies the color of the foreground text itself, the BG1 and BG2 specify the background color(s) Only BG1 is used if Banding is Off, both BG1 and BG2 are used if banding is On.

To alter a color, click the displayed color box and you will be shown the standard system color chooser dialog.

Custom Colors

Note: When you create custom colors using the color chooser, SPFLite will "remember" these custom

colors for your future use. This means you can create and save 16 custom colors.



Created with the Personal Edition of HelpNDoc: [Easy EPub and documentation editor](#)

Keyboard Customization and Keyboard Macros

Contents of Article

[Introduction](#)

[Key Mapping does not apply to Windows dialogs](#)

[Nothing can go wrong, go wrong](#)

[Start SPFLite directly into KEYMAP](#)

[Starting Over](#)

[Planning Ahead for Recovery](#)

[The KEYMAP LIST](#)

Introduction

Since most interaction with an editor is through the keyboard, the ability to customize it greatly affects your productivity. The Key Mapping features in SPFLite reflect the importance of this.

The Keyboard tab of the Options dialog provides basic on/off options to control scrolling using the arrow keys, whether the editor defaults to Insert Mode, and how Insert Mode Reset is handled. See [Options - Keyboard](#) for details.

All other keyboard customization is performed in the Keymap dialog. This dialog is reached by entering the [KEYMAP](#) primary command.

You can also use the **KEY** or **KEYS** aliases to bring up the same Keymap dialog, similar to ISPF.

From this dialog you can:

- Customize the activity that each key or key combination performs
- Assign primary commands, line commands or keyboard primitive functions to be performed by each key
- Provide optional labels to key mappings, so these can be displayed as Key Help information on the screen
- Create keyboard macros to perform a series of keyboard activities, and assign that sequence to any key or key combination
- Define the actions to be performed when the mouse buttons are pressed

As mentioned in the [Welcome to SPFLite](#), the the [KEYMAP](#) facility and mappable keyboard functions play an important role in providing many of the powerful features of SPFLite. IBM ISPF did not give as much emphasis to this, because 3270 keyboards only allowed for mapping of PF keys, whereas SPFLite can map almost anything. The time you spend learning about this facility will be well worth the effort.

Key Mapping does not apply to Windows dialogs

It is important to remember that SPFLite Key Mapping **only** applies to the File Manager and Edit/Browse screens. Such screens are under the direct control of SPFLite, and so it is able to read and interpret keyboard scan codes and mouse activity according to how you set up your Key Mapping.

When you see "dialogs" that are under the management of Windows itself, SPFLite Key Mapping does not get involved. That is because, while SPFLite has **requested** such dialogs to appear, it is Windows itself that drives and handles such dialogs. SPFLite is only "notified" when the dialog is completed. Because of that, SPFLite does not see individual keyboard and mouse actions, and is in no position to interpret them using your Key Mapping definitions.

Thus, there are "two worlds" - the world of SPFLite and its Key Mapping definitions, and the world of Windows dialogs - and these two worlds never meet.

When Windows dialogs appear, you can **only** use keys as they are understood by Windows. So, when a "Enter" is required, you **must** press the key labeled "Enter", and **not some other key mapped to (Enter)**, such as

Right-Ctrl. Copy to clipboard **must** be done by Ctrl-C, paste from clipboard **must** be done by Ctrl-V, etc. Such Windows-managed dialogs include, for example:

- The KEYMAP dialog
- The File Rename and File Delete/Purge dialogs
- All of the dialogs in SPFLite Global Options (General, File Manager, Submit, Screen, Keyboard, Mouse)
- Message Box dialogs that may have been issued by an SPFLite macro
- SUBMIT messages
- Any debugging or program-exception dialogs that might appear

Nothing can go wrong, go wrong ...

Murphy's Law suggests that at some point, somehow, you will end up with a messed up keyboard definition and need a way out. If you can still enter commands, then issue **KEYMAP**, and carefully check each suspicious key one at a time, until you find the cause of your problem. You can also see a list of every defined key mapping by using the [KEYMAP LIST](#) command, discussed below.

But what if you can't get into the Key Map? Read on.

Start SPFLite directly into KEYMAP

You can start up SPFLite using a command line option **-KEYMAP** which will bring up the KEYMAP dialog. Once you close the KEYMAP dialog, SPFLite will start up normally.

Because this is a type of "emergency startup contingency", SPFLite will not have completely finished all of its initialization steps when you get to the KEYMAP dialog. As a result, the dialog will be displayed with a default font rather than your customary editing font. Don't worry about that for now. After you finish, and when SPFLite begins normal operation, the customary editing font will appear when you issue a KEYMAP command again.

See [Starting and Ending SPFLite](#) for more information.

Starting Over

This may be the simplest way, if you're just starting the key customization process. Just delete the **SPFLite.KBD** file from your SPFLite data directory (which is normally **\My Documents\SPFLite**). When you restart SPFLite, a new keyboard definition file with the standard defaults is recreated for you. If you do this, you will lose any keyboard customization you may have done, but at least when you restart, SPFLite will be usable, and you can try doing your keyboard customization afresh.

Planning Ahead for Recovery

There seems to be two kinds of SPFLite users, as far as keyboard configuration goes:

- those who take the installation defaults, make a few minor changes, and let it go at that, and
- users that like to "go crazy", who frequently and heavily modify the KEYMAP settings (I am in that category - RH)

If you find yourself in the "go crazy" category (it's alright - we won't tell anyone !) you may wish to plan ahead in case you update your Keymap and then later wish that you hadn't. Just make sure that you have a backup copy (perhaps several) of the **SPFLite.KBD** file.

As long as you have a recovery plan, you should be ready for any eventuality. If you ever run into keyboard problems that you can't seem to fix, just copy one of your saved KBD backup files back to **SPFLite.KBD**, restart SPFLite, and you should be good to go!

The KEYMAP LIST

In order to help you examine the contents of your entire "keymap inventory", you can issue the command **KEYMAP LIST**. When you do this,

- SPFLite gathers all known information about the mapping of every key
- It takes that information and formats it into a text file
- That file is stored into the Windows clipboard
- A CLIP Edit window is opened in SPFLite to allow you to examine the contents of your KEYMAP data
- **Note:** Only **mapped** keys will be in this list. Keys whose mapping string contain nothing but **(NULL)** will not be listed

Because you are in an SPFLite-controlled edit screen when this happens, you have available to you all the editing features of SPFLite, including the ability to **FIND**, **SORT**, **EXCLUDE**, and so on.

You can create a permanent file by issuing a **SAVEAS** command, or paste the data into an application outside of SPFLite.

The data you see can be changed in any way you like; changing it will **not** affect SPFLite's KEYMAP system. Only the KEYMAP dialog can do that. So, you can safely do anything you wish to this data.

Key Mapping Overview

Contents of Article

[Introduction](#)

[Chords](#)

[Deciding on specific chords for your needs](#)

[Accessing Special Characters](#)

[Character Maps](#)

[Key Mapping Limitations](#)

[A cautionary note about key mapping](#)

Introduction

The key mapping process requires locating an unassigned keyboard key, and assigning a command string to it. You can inspect the mapping of keys individually, or you can use the [KEYMAP LIST](#) command to inspect the mapping of all defined keys.

With few exceptions, function keys, alphanumeric keys, and other keys on the keyboard are all treated and mapped in exactly the same way.

Chords

You can use nearly any key on the keyboard, and can use that key in any of 8 eight different chords. A *chord* is a regular key, optionally in combination with the Shift, Control and/or Alt key. (Sometimes, the word *chord* is meant when at least two of the three keys Shift, Control and Alt are pressed at the same time.) Once you select a physical key for mapping, the definition of every chord for that key can be mapped at the same time, since all eight shift possibilities are displayed on a single screen, as shown below.

For example, the F1 key can be assigned 8 different functions, one each for the chord combinations F1, Shift-F1, Control-F1, Alt-F1, Shift-Control-F1, Shift-Alt-F1, Control-Alt-F1 and Shift-Control-Alt-F1.

What would you *do* with all those keys? Well, nobody is likely to use them *all*, but you can use some of them for launching macros and for typing foreign letters and symbols. For example, you could map the lower-case Spanish letter **ñ** to Ctrl-N and map the upper-case Spanish letter **N** to Ctrl-Shift-N.

Deciding on specific chords for your needs

When you pick the keys to use for these purposes, the main thing is to choose something that makes sense to you, so you will remember it. You can use any strategy you like, but time and experience have shown that some choices work better than others. Here are some principles to consider:

- If you think of the Shift, Ctrl and Alt keys as "modifier" keys, then the best approach is to use the fewest modifier keys you can to achieve your purpose. The more modifier keys you pick, the harder it is to type.
- When modifier keys are close together, or are aligned straight across from each other (horizontally or vertical) they are easier to use than when they are on a diagonal from each other. That means, for two modifiers, the closest two keys are usually Ctrl plus Shift. Next closest are Ctrl plus Alt. Usually on a diagonal and harder to type are Alt plus Shift. Your specific keyboard layout or your personal preference may affect what is easier for you to use. Using all three modifiers is rare, because it is the hardest to type and can only be done with two hands, and would normally only be used for unusual, seldom-required functions.
- Common usage dictates that the Shift key should be included where alphabetic data is involved.

- The Ctrl keys (especially the Left Ctrl key) are normally at the lower corners of the keyboard where they are easy to find quickly. The Alt keys are often less accessible and are frequently next to special Windows keys and/or Fn keys on a laptop, so Alt can be a little harder to get to. In a choice between Ctrl and Alt, Ctrl is often better.
- Based on the principles above, and customary keyboard layouts, the usage of modifier keys should generally be arranged wherever possible in this order of priority (from most-preferred to least-preferred):
 - Shift
 - Ctrl
 - Alt
 - Ctrl-Shift
 - Ctrl-Alt
 - Alt-Shift
 - Ctrl-Alt-Shift

In the example above, we mapped the lower-case Spanish letter ñ to Ctrl-N and the upper-case Spanish letter Ñ to Ctrl-Shift-N. Another reasonable choice would be Alt-N and Alt-Shift-N (although Alt-Shift is harder to type than Ctrl-Shift).

However, a choice like Ctrl-N and Ctrl-Alt-N could be confusing, since it's not clear which chord is used for the upper case Ñ. We picked Ctrl-N and Ctrl-Shift-N because it retains the use of the shift key for a letter (rule 3) and because the Ctrl/Shift combination is usually the closest together and so is the easiest to type (rule 2).

Bear in mind that, on international keyboards that possess an **AltGr** key, they are designed to transmit the same keyboard data that would occur if you manually held down the **Ctrl** and **Alt** keys at the same time. That means, for SPFLite purposes, the single key **AltGr** and the two keys **Ctrl** and **Alt** used together, mean exactly the same thing. By default, SPFLite maps all keys with a printable value (the alpha-numeric and special character keys) so that they will function normally with the **AltGr** key.

Accessing Special Characters

But wait a minute! If I don't *have* a certain special character mapped to any key, how will I ever *get* it into the key mappings I want?

If you happen to know the decimal number of the character you want, you can enter it using ALT+0nnn. For example, you can enter an EBCDIC ¬ not sign (X'AC') by holding down the ALT key and typing 0172, where X'AC' = decimal 172.

You have to actually enter the leading **0**, or otherwise you get an OEM key value instead of an Ansi value, and for keys that are >= X'80' they won't be right. SPFLite uses Ansi encoding, not OEM encoding, and so 172 and 0172 are **not** the same character.

This ALT + number technique **only** works in the KEYMAP GUI entry fields, **not** in the main SPFLite edit window. That's because SPFLite's extensive key mapping logic would capture any special keys like ALT + a number, and Windows would never see it to convert the number to a character. When you are within a GUI dialog, Windows itself directly manages both the screen and the keyboard input instead of SPFLite, and that's why this technique works. However, once you *have* a key mapped to some special character, you can use it on the SPFLite editor screen just fine.

The easiest way to get special characters is to use a character map.

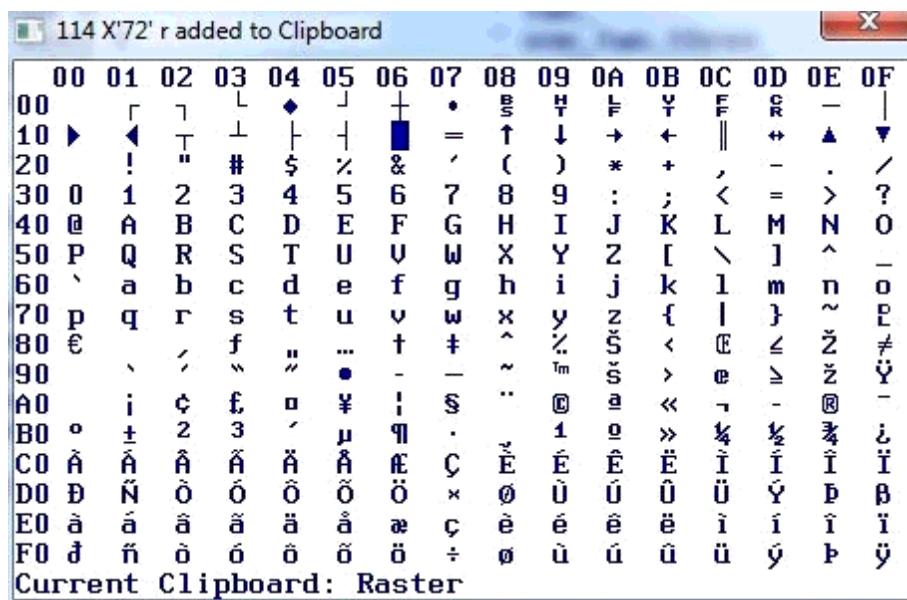
Character Maps

If you want a graphical way to do this you can bring up the Windows Character Map utility, click on the desired letter, and copy it to the clipboard. Then come back to SPFLite and paste it into the key mapping definition you need.

SPFLite really works internally only with Ansi characters, so even though the Windows Character Map utility can select many different Unicode characters, you can only use Ansi in the key mapping. You will see on the lower-left of this utility a Unicode hex value for each character. As long as it shows U+00FF or lower, you can use it.

If you find that you need certain special characters in many places, including outside of SPFLite, you can select a foreign or alternative keyboard layout, or create a custom layout using a utility program such as **Microsoft Keyboard Layout Creator** or **KbdEdit**. If you do this, you will not need to map any special characters in your layout with SPFLite's KEYMAP. (An Internet search will find this packages.)

SPFLite now also supports its own character map. In the Key Map display, there is a button on the lower right corner of the screen, "Show Character map". If you click on this button, you will see a pop-up display with a 16 by 16 grid of characters, something like this:



An advantage of this over the Windows character map, is that SPFLite uses the same font for this as is used for your edit screen; so it better reflects what you will really **see** for the chosen character(s). The specific characters available and their appearance depend on the display font you have chosen; the pop-up above shows the RASTER font, available from SPFLite as a download.

Alternate Character Sets

The character display shown by SPFLite is sensitive to the character set you have specified via the **SOURCE** setting. For example, if you display the character map while working on an EBCDIC file, the display will show the EBCDIC character set, and will place appropriate characters into the clipboard based on the in-use character set. The display shown in may be in another collating sequence (either EBCDIC or user-defined), and will report characters placed into the Clipboard based on their collating sequence. Storing an ASCII zero will be reported as X'30' while an EBCDIC zero will be reported as X'F0'.

Once this display appears, you place characters into the clipboard as follows:

Using the left mouse button, click on any desired character in the grid. Any prior contents of the clipboard are erased and replaced by the single character just clicked.

Using the right mouse button, click on any desired character in the grid. The prior contents of the clipboard are retained, and the single character just clicked is appended to the end of the clipboard.

Note that this same display can be brought up and used in an edit session by means of the keyboard primitive function [\(CharSet\)](#), which you would have to map to a key of your choice. For example, suppose you mapped Ctrl-Shift-A to [\(CharSet\)](#). Then, while editing a file, if you needed a special character inserted, press Ctrl-Shift-A, and this window will appear. You then place the desired key(s) into the clipboard, dismiss the window by clicking

on the X, and then paste the character(s) into the desired location, which for most users would be done by pressing Ctrl-V (or using the right mouse button, if you have configured your mouse to operate that way).

Finally, you can use SPFLite itself to create special characters by editing in HEX mode. Once you have composed the characters you want, set HEX OFF, and then using the mouse or cursor keys, highlight the desired text and copy it into the clipboard.

Key Mapping Limitations

As mentioned previously, not just function keys can be mapped, but almost *any* key can be user-customized. Which keys *cannot* be mapped? An obvious one is Ctrl-Alt-Delete. Any key sequence that Windows has reserved for its own uses can't be mapped. SPFLite has marked *some* of these as **This entry is Reserved, Do NOT use**. Even if SPFLite *itself* doesn't mark or prevent you from mapping a reserved key, your system may still prevent its use. If you are curious about which key combinations may be reserved, one description can be found at the web site <http://windows.microsoft.com/en-US/windows7/Keyboard-shortcuts>. This list of reserved keys is sensitive to the version of Windows you are running, as well as to optional Windows run-time parameters you may have set, such as Accessibility shortcuts.

Some key sequences used in Microsoft software are merely *conventions*, and strictly speaking, are not "reserved". For example, Microsoft uses Ctrl-C for "copy" and Ctrl-V for "paste", but these are just conventions; they are not reserved. You can map Ctrl-C and Ctrl-V in SPFLite to do anything you wish.

These conventions still apply to any "control" or "dialog" or other GUI displayed by Windows itself, where keys like Ctrl-C and Ctrl-V mean what Windows says they mean, and nothing else.

There are a few basic keys that cannot be mapped. These are the Shift keys, the Windows keys (the ones with the "flag" on them), and the Caps Lock key. (However, the Windows *Application* key is fully mappable. In the picture below, the Application key is just to the left of the right Ctrl key. If you click on it, the name for the key will display as APPMENU.)

Some keys have only limited mapping, so that *partial* mapping is permitted, but **not** with all 8 possible combinations of Shift, Ctrl and Alt. These limited keys are Escape, Scroll Lock, Tab, and the Ctrl and Alt keys themselves. (Users of SPF-style editors on the PC will often map the right Ctrl key to [\(Enter\)](#) in SPFLite, since it simulates the *Enter* key on a 3270 terminal.) When you click on a key for which only partial mapping is available, the chord combinations you are not allowed to use will be grayed-out.

Some keys like Alt-Tab are reserved to swap between various Windows applications that are running at the same time. If SPFLite mapped this key, you'd never be able to Alt-Tab **out** of SPFLite to do something else. So, even if it were technically possible to "grab" these keys (which, in some cases, it is), it wouldn't be a good idea. SPFLite is already mapping as many keys as it can, and it's not likely to be able to increase the list of mappable keys beyond what it's doing now. (Believe me, I've already asked George to support everything humanly possible right now - RH)

By the way, you should be aware that you can map the **Pause** key as well. Most software totally ignores this key. This might open up some useful possibilities for a command you used frequently and wanted to dedicate to a specific key.

A cautionary note about key mapping

As you are about to see in the next section, key mapping can be quite involved, powerful, and even fun - **if** you enjoy maximizing its features. If you are a good typist with good dexterity, you can greatly benefit from what SPFLite's key mapping has to offer.

However, if you are **not** that great a typist, at times you could find yourself inadvertently hitting the wrong keys or key chords, more often than you'd like. If you load up many different keys with many "shift combinations" like Ctrl-Shift, Ctrl-Alt and Alt-Shift, etc. you could end up performing editing actions by accident that you didn't intend, possibly ones that are hard to correct or undo. This is especially true if you are what is colorfully called "fat-

fingered," or if you type a lot while your hands (or the rest of you) are tired. You could also end up with so many key definitions that you start forgetting what they are.

If that describes you, overloading the key mapping system with a very large number of key definitions may not be a wise choice, and could end up making the use of SPFLite a more frustrating experience than you would prefer.

So, the cautionary note is to 'know yourself' as to whether making your keyboard setup very complicated is a good idea or not. Don't get so carried away that you make things worse instead of better!

Using the KEYMAP Dialog

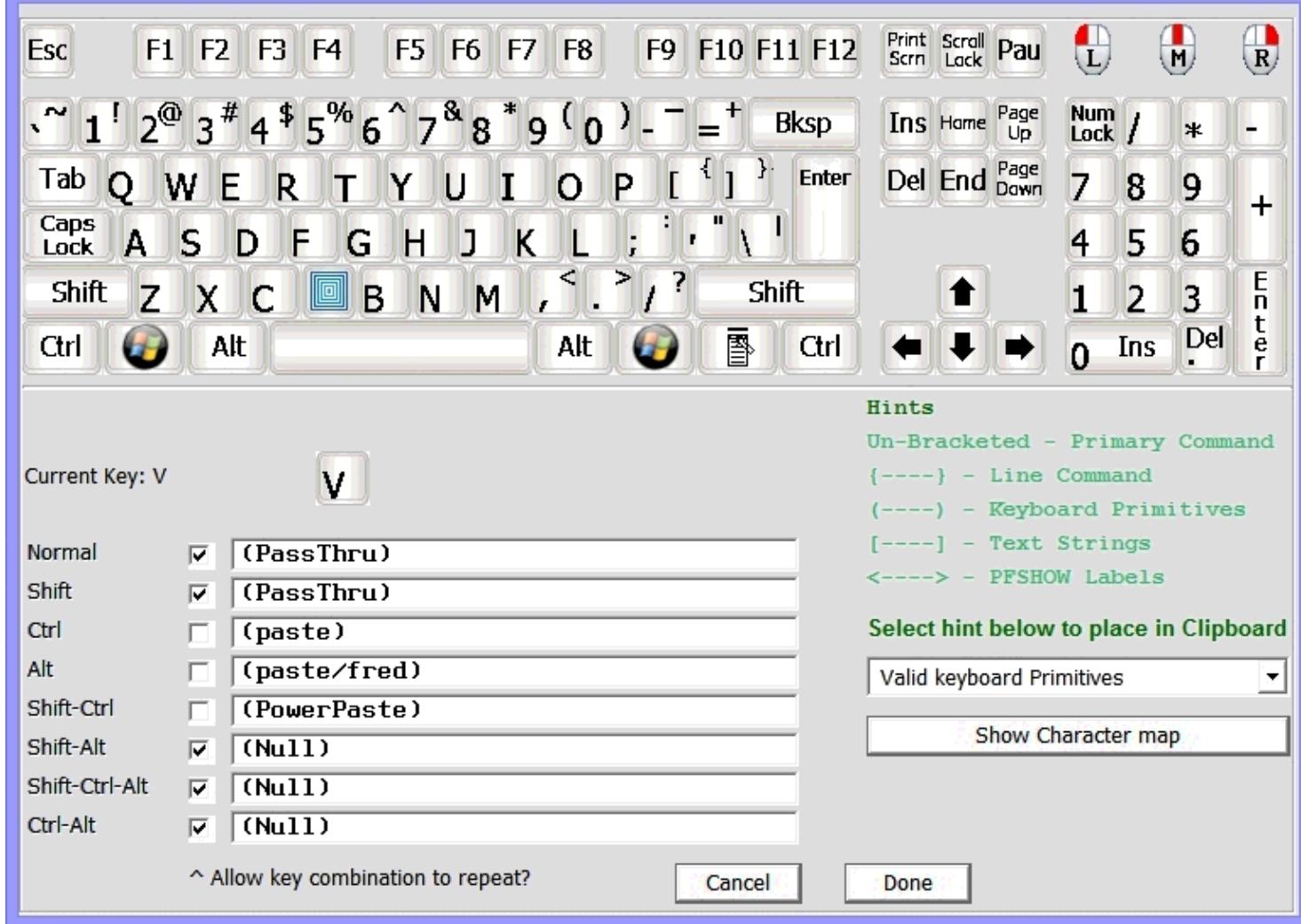
Contents of Article

- [Introduction](#)
- [Non-US 104 Key Keyboards](#)
- [Using the Keymap Dialog](#)
- [A Note about Line Command Mapping](#)
- [Customizing a Single Key](#)
- [Entry of key mapping definitions](#)
- [Dealing with pasting errors in KEYMAP](#)
- [Overriding \(Passthru\) Mappings](#)
- [Available key mapping definition options](#)
- [Assigning Multiple Functions](#)
- [Location of Primary Commands in a Macro](#)
- [Auto Repeat / Set Cursor Position](#)
- [Controlling the Keyboard Repeat Rate](#)
- [Finalizing Your Keymap Customization](#)

Introduction

The SPFLite Key Mapping Dialog appears as follows:

SPFLite KeyMap



The Keymap dialog contains two basic areas.

The top half contains a map of a typical 104 key keyboard. Although layouts may differ on your computer, especially for laptop keyboards, the customization is performed via the logical key *name*, not its physical location. So regardless of *where* a key is on your keyboard, customization is done using its *name*. Simply use the equivalent key on the provided layout.

In the upper right hand corner, you will also see three icons representing the 3 mouse buttons. That is because these buttons are now defined in the same way as keyboard keys. (After all, that's what they are being treated like, in this function). The L, M and R refers to the Left, Middle and Right mouse buttons.

The definitions for each key are displayed in the same font as you choose for the edit session. So, if you have mapped some special-character keys, they will display here exactly as you see them in an edit session or in the [\(CharSet\)](#) popup window.

Because these are *logical* key definitions, SPFLite has no idea what your *physical* keyboard layout really looks like. For laptops, you may not have all these keys; there may be only one Windows key instead of two. You also may not have a physical numeric pad, but may be able to generate those keys by holding down an auxiliary key, often labeled **Fn**. For example, **Fn** + the *regular* 7 key may be the same as the *keypad* 7 key.

This means it is possible you can define a key mapping for a key that doesn't exist on your keyboard. No harm is done, but you just wouldn't be able to use it. Consult your hardware documentation for details on your particular keyboard. SPFLite is unable to map auxiliary keys such as **Fn**, or any extra keys such as multimedia keys, that are outside the realm of a standard 104 keyboard.

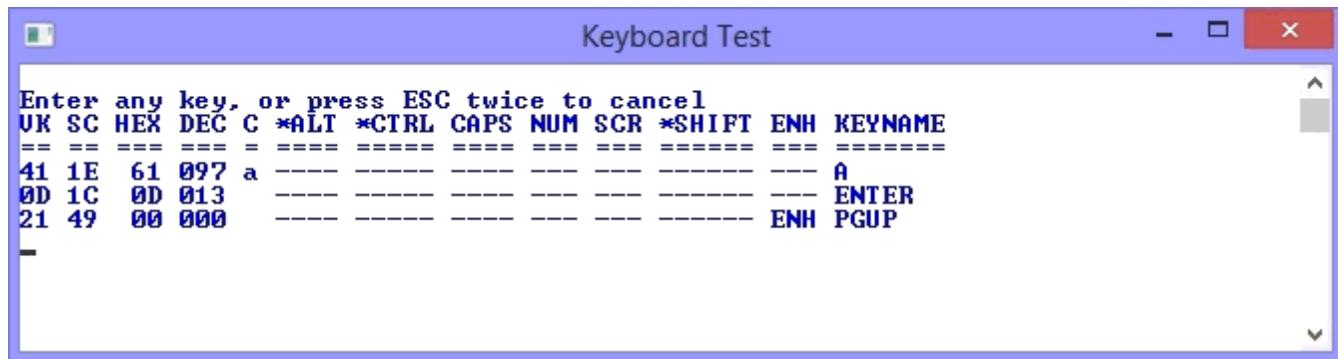
Non-US 104 Key Keyboards

Some SPFLite users do not use a conventional US/English keyboard but instead use a national keyboard which contains additional keys required by other languages. Often, these keys require the use of the **AltGr** key (the right-hand Alt key, which stands for Alternate Graphic). As well, the keyboard layouts differ, and there may be more or fewer keys than the layout shown above. This can make SPFLite keyboard customization a challenge.

Note: It's important to bear in mind that SPFLite internally processes data using ANSI characters only, which are stored in the Microsoft 1252 character set. MS 1252 supports the Western European languages: Afrikaans, Basque, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Icelandic, Indonesian, Italian, Malay, Norwegian, Portuguese, Spanish, Swahili, and Swedish. That's a lot, but it's not everything. If you need to type something in a language other than those, SPFLite may not be your best choice.

These issues are assisted by:

- First, SPFLite automatically maps all alpha-numeric and special character keys which could use the **AltGr** key to create national characters. These should all work correctly 'out of the box'. They all have their **Ctrl-Alt** value set to [\(Passthru\)](#) instead of the normal [\(Null\)](#). The [\(Passthru\)](#) setting allows the key to perform its Windows-defined default action. (This technique works because, on keyboards that possess an **AltGr** key, they are designed to transmit the same keyboard data that would occur if you manually held down the **Ctrl** and **Alt** keys at the same time. So, for SPFLite purposes, the single key **AltGr** and the two keys **Ctrl** and **Alt** used together, mean exactly the same thing.)
- Second, to assist in identifying exactly which key on your physical keyboard is equivalent to which key on SPFLite's KeyMap dialog, a utility program has been included in the distribution. The link to it is in the SPFLite Start Menu folder; look for the **KeyboardTest** entry. When clicked, this utility displays a console screen where you can type keys. For each key you press, it will display detailed keyboard status information. The right-hand entry on each line will identify the SPFLite key name which matches your physical key. The display looks like this:



The column headed **KEYNAME** is the one telling you which SPFLite key needs to be customized. In the sample shown, the keys pressed were the **A** key, the **Enter** key, and the **PageUp** key. The **KEYNAME** string, like **PGUP**, corresponds to the name displayed in **KEYMAP** where you see the label **Current Key** for a given key.

Some keyboards have extra keys that are not used by SPFLite, and the **KeyboardTest** tool will report these keys as **Unknown key**. If this happens, you can use those keys in SPFLite for their original purpose, the same as a [\(Passthru\)](#) key is used, but you will be unable to remap these unknown keys. That is because SPFLite does not know what its "keyboard scan code" means, and doesn't know what to do with it, except to "pass it through".

The very fact that you can't map them means that you can't even map them to [\(Passthru\)](#). The point is that SPFLite will pass-through any key codes it doesn't recognize as if they were mapped to [\(Passthru\)](#). You don't have to do anything in SPFLite to use those keys - just use them.

If you have a multi-media keyboard with keys for things like Play, Pause and volume controls, SPFLite never sees these keys, so you don't need to worry about them.

Using the Keymap Dialog

The bottom half of the KeyMap dialog becomes active once you click on an individual key icon in the top half. In the screen display above, the **V** key has been selected. Note the **V** key graphic in the top half has been obscured, and the key is then displayed in the bottom half, along with the current action assignments for each of the eight possible chord combinations.

The right side of the bottom half provides reminders of the format of various command types you can enter in the key mapping definition fields.

Also, there is a button labeled **Show Character Map**. When clicked, a new separate dialog will be opened, with a character map that uses the ANSI collating sequence. See [Character Maps](#) for more information.

At the bottom are two buttons **Cancel** and **Done**. Clicking **Cancel** will terminate the dialog without saving any changed keyboard assignments. Clicking the **Done** button will save the changes and return to your edit session. Saved changes to the key map will take effect immediately; no restart of SPFLite is required to active them.

Within the Key Map window, you can jump from one of the eight various definitions (all the combinations of Shift, Ctrl and/or Alt) of a key to another by using the Tab and Backtab keys. This reduces the need to frequently alternate between the mouse and keyboard when entering many definitions for the same physical key.

A Note about Line Command Mapping

The **:** colon notation for key-mapping of line commands that is utilized in IBM ISPF is not directly supported. In prior versions, if you tried it, you would have created invalid syntax. For example, if you previously mapped **Alt-R** to just **:R**, when you use it, it would **not** repeat a line but would simply report, **Unknown command**.

Now, an attempt to define a key with a line command like **:R** will result in a warning pop-up message advising the user that the command has been automatically converted to the SPFLite-compatible equivalent format of **{R}**.

Customizing a Single Key

To customize a key, perform the following steps:

- On the command line enter the **KEYMAP** command and press Enter. The Keymap dialog (as shown above) will be displayed. (**KEY** and **KEYS** are aliases for **KEYMAP**.)
- On the keyboard graphic display, click on the key to be customized. The key will be obscured to indicate it is active, and the current settings for that key will be displayed in the lower half of the dialog.
- Click in the text box next to the particular chord (Normal, Shift, Ctrl, etc.) of the key you wish to alter.
- Enter your desired function definition.

Entry of key mapping definitions

You can enter key mapping definitions manually, or by pasting them in. You will see the text of each key mapping definition is displayed in the same font as your fixed-width editing font. (If you start up SPFLite with the **-KEYMAP** command line option, you will temporarily see the dialog displayed with a default font.)

To assist in remembering primitive function names, a drop-down list is provided in the lower right corner of the dialog, shown as **Valid keyboard Primitives**. If you open this list and select an item by clicking on it, the value will be placed in the clipboard. You can then paste that value into the text box for your desired key, using the **Ctrl-V** key. This saves time and avoids typing errors.

You can also paste any text into a definition field it you obtained it from outside the KEYMAP dialog, or even

outside of SPFLite. It is your responsibility to ensure that such pasted mapping definitions are properly formatted. After you paste data, you can still manually modify or correct it as needed.

Dealing with pasting errors in KEYMAP

If you were to inadvertently paste some very large amount of text into a key mapping field by mistake (say, if you forgot that you just placed a large amount of data into the Windows clipboard previously), you can press Ctrl-Z right away, which will 'undo' the paste operation and will erase the key definition field you were just working on.

To limit the effect of an inappropriate paste into one of these fields, if the clipboard contains multiple lines of text, only the first line will get pasted into a key mapping field.

Overriding (Passthru) Mappings

Ordinarily, key mappings that default to (Passthru) are left alone. But, you may have a requirement to override them. What sort of situations might call for this? Here are some possible examples:

- You have a non-US keyboard and some of the characters are in different locations
- Perhaps you want to try a non-QWERTY layout, like DVORAK
- You want to 'invert' the shift-sense of a key. For example, let's say you use SPFLite mostly to write programs in a language that uses underscores a lot (like C) but you don't use the minus sign all that much. Suppose you wanted the minus/underscore key to produce underscores by itself and minus when shifted. You could invert the minus and underscore by putting a mapping of `[_]` for the unshifted mapping and `[-]` for the shifted one.

Available key mapping definition options

(Passthru) [\(Passthru\)](#) is a primitive keyboard function which simply says the key should “pass through” whatever Windows would normally have done when the key is pressed. This is the standard value for most of the alphanumeric keys, and provides the most obvious key assignment. So, for instance, you don't have to spell out that the plain **X** key types “X” – you just “pass it through”.

If you attempt to completely blank-out or delete a key definition for a Normal or Shift form, SPFLite will replace it with [\(Passthru\)](#). For other keys, erasing a definition will cause it to become [\(Null\)](#).

For the mouse *only*, [\(Passthru\)](#) has no meaning and is treated the same as [\(Null\)](#).

(Null) [\(Null\)](#) requests that no action be performed. This is the value for unassigned key combinations.

If you attempt to completely blank-out or delete a key definition (other than the Normal or Shift forms), SPFLite will replace it with [\(Null\)](#).

When you have a key mapped to a line command, there is an implied [\(Enter\)](#) that is automatically inserted. In some cases, this implied [\(Enter\)](#) may not be what you wish. To suppress the implied [\(Enter\)](#), you can put [\(Null\)](#) after all other commands or functions.

command Any text entered **without** being enclosed in `()`, `[]`, `{}` or `<>` characters is assumed to be an SPFLite primary command and its operands, and it will be entered as such on the command line. Such commands are processed **as if** there was an implied (Enter) function on the command.

When text is entered on the command line, and a command key is pressed rather than the ENTER key, SPFLite concatenates the contents of the command line to the definition of the

command key. The result is handled as a single, composite command by SPFLite.

For example, when you use a Command key defined as a scroll command (**UP**, **DOWN**, **LEFT**, or **RIGHT**), the value entered on the command line becomes the **operand** of the scroll command. Assume you have the **PageDn** key mapped to the **DOWN** primary command. Entering **HALF** on the command line and then pressing the **PageDn** key results in a composite command of **DOWN HALF** being issued to scroll the screen down by half its height.

You may choose to override this concatenation when you set up the command key definition by preceding the command with a ! (exclamation mark character). This causes the programmed command to **replace** whatever is in the command line. e.g. If you have a key programmed for **RESET**, it would probably be wise to prefix it with a ! to prevent the **RESET** command processor being confused by possible command line 'remnants' existing when the command key is pressed.

Note: As mentioned above, a plain, unenclosed command is treated **as if** there is an implied (Enter) function on the command. That works well if the only thing you want to do is a primary command. However, you might wish to combine line commands, primary commands and other functions in a single key mapping. To do that, you will need to explicitly specify each part. For the part of the mapping that has a primary command, you will have to use all the steps you would take if you did it manually, and the command itself becomes a text string in brackets. So, an implicit primary command might be **CHANGE ABC DEF**. If you wrote this as an explicit command as part of a larger key mapping string, the part for the primary command would be mapped as

(Home)[CHANGE ABC DEF](Enter)

{line-cmd}

A string enclosed in { } braces is treated as a *line command*. The text of the string will be entered in the line command area of the line on which the cursor is currently located. If the cursor is not currently on a valid line, then no action is performed.

In ISPF, a PF key might have assigned the value of :**I2** to request an insertion of two lines, a notation which SPFLite does not support. That request in SPFLite is entered as **{I2}**. If you attempt to define a line command using the ISPF : colon notation, SPFLite will automatically convert it to the brace notation and will issue a warning.

Line Command Toggling: If you have a key mapped to a line command like **{CC}**, and there is any existing line command **other than CC** in the sequence area, the **CC** command will be entered there, replacing the existing command. If the sequence area **already** has a **CC** line command, and you press the key mapped to **{CC}**, the existing **CC** line command will be erased. This line-command toggling action allows you to easily undo a line command if you placed it on a line by mistake.

You can toggle the state of excluded lines and User Lines with the **TX** and **TU** line commands, respectively. **Suggested** mappings that retain the current cursor location are as follows:

Ctrl-Alt X = **{TX}(Enter)(Up)**
Ctrl-Alt U = **{TU}(Enter)(Up)**

Implied (Enter): When you have a key mapped to a line command such as **{CC}**, there is an implied (Enter) that is automatically inserted afterwards. This implied (Enter) may or may not be what you wish. If you don't want this implied (Enter), there are some ways to address this:

- To suppress the implied (Enter), you can put **(Null)** after all other commands or functions,

like **{CC}(Null)**.

- In IBM ISPF, a mapped line command will leave the cursor in the first position of the sequence area. You can replicate this action with **{CC}(LineNo)**.
- As may be seen above in the **TX/TU** example, line commands which you need to have acted-upon right away **cannot** have the implied **(Enter)** suppressed; otherwise, nothing will happen.

Special Line Commands: There are special line-command formats for labels and tags. Note that labels and tags can be set, cleared or toggled independently of each other and can coexist on the same line.

{. label}	Enter <i>label</i> into sequence area
{.. label}	Toggle <i>label</i> ; enter <i>label</i> if no label present, else clear any label that is present
{.}	Clear any label that may be present
{..}	Clear any label that may be present; same as {.}
{: tag}	Enter <i>tag</i> into sequence area
{:: tag}	Toggle <i>tag</i> ; enter <i>tag</i> if no tag present, else clear any tag that is present
{:}	Clear any tag that may be present
{::}	Clear any tag that may be present; same as {:}

(primitive)

Just as with **(PassThru)** and **(Null)** above, any string enclosed in **()** parentheses is treated as a keyboard primitive function. Keyboard primitives are those functions *other than* line commands or primary commands. There are currently over 90 keyboard primitive functions available. Only primitive names predefined by SPFLite may be specified.

To assist in remembering primitive function names, a drop-down list is provided in the lower right corner of the dialog, shown as **Valid Keyboard Primitives**. If you open this list and select an item by clicking on it, the value will be placed in the clipboard. You can then paste that value into the text box for your desired key, using the **Ctrl-V** key. This saves time and avoids typing errors.

A description of all primitive functions is provided in the "[List of Keyboard Primitives](#)"

**(primitive/
ClipName)**

For primitive functions which refer to the clipboard, the default is always the normal Windows clipboard. If you wish to direct the function to use a Private Named Clipboard, the clipboard name should be provided following the primitive name, separated by a **/** character. For example to perform a Copy operation to named clipboard *Fred*, you would enter **(COPY/FRED)** for the key value. To perform a Paste operation from named clipboard *Bill*, you would enter **(PASTE/BILL)** for the key value.

**(cursorFunction/
count)** These five specific cursor movement functions only will accept a **count** option. These functions are:

- **(Left/count)**
- **(Right/count)**
- **(Up/count)**
- **(Down/count)**
- **(Column/count)**

The *count* field causes the given function to move the cursor left, right, up or down the indicated number of columns or lines. For Column, *count* indicates the absolute column number on the current data line where the cursor is to be positioned. For these functions, the

count field is similar to the **repetition factor** described next, except that the performance in repositioning the cursor is **much** faster.

If *count* and the preceding slash are omitted, it is the same as if the value were **1**.

(n:primitive) You can also specify a **repetition factor** for primitive functions. The repetition factor is specified as a decimal number just after the opening left parenthesis of the function, and followed by a colon.

For example, the function [\(Right\)](#) will move the cursor one character to the right. If you wanted to move the cursor four positions to the right, you could enter this as **(Right)(Right)(Right)(Right)**.

With a repetition factor, this can be simplified to **(4:Right)**.

Repetition factors can only be used within primitive functions in **()** parentheses, not on un-enclosed primary commands or on the other types of enclosed keyboard items like **[]**, **{ }** or **< >**.

A description of all primitive functions is provided in the "[List of Keyboard Primitives](#)"

[text-string] Any text string enclosed in **[]** brackets will be entered at the current cursor location as if manually typed from the keyboard. This could be used, for example, to enter boilerplate text, or any other repetitive phrase. Embedded blanks are allowed in the text string. You could map a key to a foreign letter by putting that one letter in brackets, such as the letter **[Ø]**.

Note: Former Tritus SPF users that were able to map a string like **ABC** using a primary command of **DATAABC** can achieve the same thing with an SPFLite key mapping definition of **[ABC]**.

What if your data actually *contains* a left or right bracket? Just double any brackets you need to use as literal values.

Example: You want a key to generate:

x[y] = z

Solution: In the key definition field, you enter:

[x[[y]] = z]

(nn:[text-string]) This variation of the normal text string enclosed in **[]** brackets is used when you desire a repetition of the provided string. In addition to being a simplified format, and allowing you to define long strings that would be awkward and error-prone to type manually, this method is also significantly faster in performance. This could be used, for example, to enter a long string of equal characters.

Example: You want a key to generate a string of 40 * characters, so rather than entering:

[***...*****]**

you enter this instead:

(40 : [*])

Similarly if you enter:

(4 : [----+----|])

it would be the same as

[----+----|----+----|----+----|----+----|]

<key-label> Text enclosed in **<>** angle brackets is a *key label*. If present, the optional **<key-label>** field

must be **leftmost** in the key definition field. A key label provides the text that will appear, along with the key name, at the bottom of the edit screen when the display of 1 or more Keyboard Help lines is enabled. (This is like the **PFSHOW** command in prior versions of SPFLite.) See [Options - Screen](#) for activating this option. For example, if the entry for the unshifted **F3** key were coded as **<End>end** then it would show on the bottom of the Edit screen as **F3=End**.

Note that what is shown as Keyboard Help is the key **label**, not the key **definition**. So, the label of the key that is displayed, and the definition of the key that is acted on, are completely different things.

Key labels are helpful if you (or some other user you are assisting) need to be reminded what certain keys do. Once you can remember the definitions, the display of the key labels can be removed to save screen space.

Assigning Multiple Functions

Multiple keyboard functions can be entered for one key. In fact, this is how keyboard macros are created, by "stringing together" a series of keyboard functions. (This is further discussed below under "Keyboard Macros".) When this is done, the functions are processed left to right. Suppose you wanted to put a label of **.AA** on the current line, then home the cursor, and issue a **CHANGE** command against that labeled line. Say you wanted to call this activity the "ABC" key. (When you perform line and primary commands like this "the hard way" you may need an intermediate Enter after the part that sets the line label.) The following key entry could accomplish this:

```
<ABC>{ .AA} (Enter) (Home) [CHANGE ABC DEF ALL .AA] (Enter) -- Version 1
```

However, there is no need to be so detailed. Since strings outside of any enclosures like **()**, **[]**, **{}** and **<>** represent primary commands, you can just enter the primary-command as-is, if there's only one of them. Also, there is an implied [\(Enter\)](#) that takes place when line-command entries like **{.AA}** are used. That will eliminate the need for the [\(Home\)](#) and [\(Enter\)](#) functions above. The simplified, and faster version, looks like this:

```
<ABC>{ .AA} CHANGE ABC DEF ALL .AA -- Version 2
```

It is important to understand that the two macro definitions are equivalent. The reason it's important is that if you use the Keyboard Recording feature and type your keystrokes in, they will be recorded literally, much like what is seen in Version 1 above. You may wish to simplify it to something more like Version 2, for performance reasons.

Location of Primary Commands in a Macro

When a macro contains a primary command, as the previous topic discussed, performance is improved when the command is **NOT** enclosed in **[]** characters. However, if the primary command is **not** the last action in the macro sequence, it should be entered as Version 1 above.

For example, take a macro that is supposed to do a **PASTE** command after the data line where the cursor is currently on, and keep the cursor from losing its current location. The macro should seemingly look like:

```
(SaveCursor) {A} PASTE (RestoreCursor)
```

But it would actually fail, as the characters **PASTE (RestoreCursor)** would be entered in the line-command area, which won't work. After getting the **A** line command where it's needed, we need to reposition the cursor up to the Home position. The macro must be coded as:

```
(SaveCursor) {A} (Home) [PASTE] (Enter) (RestoreCursor)
```

which **would** execute as we desire.

Auto Repeat / Set Cursor Position

This is a dual purpose field, so we will treat each separately.

- **When a Keyboard Key is selected.** This checkbox next to each key entry specifies whether a particular key combination to **automatically repeat** if that key is *held down* instead of being “just typed” quickly. For most normal typing keys, auto-repeat is desirable, because it makes the keyboard faster and more responsive. However, for some kinds of key mappings, auto-repeat could be a little **too** responsive for your own good. For example, it is common to have the primary command **END** assigned to F3. Let’s say you had multiple tabs open for several files, and you intended to save and close of just *one* of them, using F3. If you inadvertently held down your F3 key a little too long you might end up closing several or **all** your edits before you were really done with them. For keys mapped to powerful commands like **END**, you should clear the checkbox, so that auto-repeat is disabled for that key. That way, keys with powerful commands that you want to carefully apply one at a time don’t suddenly get repeated many times by mistake.
- **When a Mouse Key is selected.** This checkbox specifies whether, before any command action is performed, the **cursor should be moved** to the character location on the screen where the click occurred. Depending on the particular command you are going to issue, you may or may not want the cursor moved from its current location. e.g. If all you wish a mouse button to do is move the cursor, you would tick this checkbox and leave the command as **(Null)**. Note that mouse buttons don’t have an auto-repeat checkbox because mouse buttons do not auto-repeat when held down.

Controlling the Keyboard Repeat Rate

SPFLite respects the keyboard timing options that are configured in Windows. If you select the Windows Control Panel and then Keyboard Properties, you can set the keyboard Repeat Rate and Repeat Delay.

If you find that your keys are “jumping the gun” and repeating too soon when you hold them down, increasing the Repeat Delay may help. However, for most users, you will get the greatest productivity by making the keyboard operate as fast as possible.

Finalizing Your Keymap Customization

Once you are finished with your mapping activities (and you may customize as many keys at once as you like), click the *Done* button at the bottom. You don’t have to “save” each key one at a time; *Done* will save every key you have defined or modified.

Keyboard Macros

Contents of Article

[Introduction](#)

[Creating a macro using the Record function](#)

[Keyboard recording and Power Typing](#)

[Saving/Editing key contenta as .DO Files](#)

[Case Study: Implementing bookmarks](#)

[Case Study: Making editing easier on laptops](#)

Introduction

SPFLite keyboard support allows you to create keyboard macros.

Note that Keyboard Macros and Command Macros are completely different facilities. Refer to [Command Macro Support](#) for more information.

A keyboard macro is a predefined series of keystrokes that are 'played back' when a specific key is pressed. You can create this series of keyboard entries manually, as described in [Customizing a Single Key](#), by typing each primitive function name, text, line command etc. from left to right in the appropriate text box.

For macros with only a few entries, this is certainly feasible. For longer macros, the best approach may be to use the keyboard recording facility. To do this, you must first know what the macro recording key is mapped to, or define it if necessary.

Note that the keyboard recording facility is not 'magic'. You will see that the end result of recording a macro is a text string in a key mapping definition entry, one that you could have typed yourself 'the hard way'. The benefit of keyboard recording is that you can define a macro by concentrating on the actual keystrokes you use, rather than the syntax of the various entries described above.

If you have not redefined it, the default for the keyboard [\(Record\)](#) function is the Scroll Lock key. You are free to assign the [\(Record\)](#) function to any key you wish, as described above. For laptop keyboards, you may need to hold the **Fn** key to access the Scroll Lock key.

It may be best to not initially change the definition of the [\(Record\)](#) function, so as not to confuse yourself while reading this documentation.

Creating a macro using the Record function

To record a keyboard macro, follow the following steps:

- Decide what key or key combination you will be assigning the macro to.
- Establish a normal edit session, and position the data and current cursor location exactly as you expect it to be when the macro is eventually used. If the macro is going to initially move the cursor to a common position (like Home), then the current cursor position may be unimportant.
- Press the key to which [\(Record\)](#) is assigned. By default, this is set up as the Scroll Lock key.
- You should see a red box in the status line saying **KB Recording** to indicate that Keyboard Recording is taking place.
- At this point, carefully perform the keyboard activity that is to become the macro. This can consist of any combination of text entry, cursor movement, line command and/or primary command you desire. (If the sequence is complicated, you may wish to perform a "dry run" first to make sure you have it "just right" and write down the exact steps you will need.)
- When all keyboard activity that is to be part of the macro is complete, press the key to which [\(Record\)](#) is

assigned **again**. That will finish the Keyboard Recording, and the **KB Recording** status indication will disappear.

- The required text string which makes up the macro will be placed in the Clipboard,
- Enter a **KEYMAP** command to bring up the Keypad dialog.
- Click the key you chose in the first step above.
- Click the text box next to the particular key you wish to use, with any combination of Shift, Ctrl, and/or Alt.
- Erase the current contents of the text box.
- Right click in the box and Paste in the macro from the Clipboard.
- Click the *Done* button.

Your macro is now ready to use.

You should test your macro after defining it, to make sure it does what you intended. Sometimes macros recorded as keystrokes this way are a little tricky to enter correctly on the first try, especially if they are long and complicated. It might take a few attempts to get every keystroke just right. If you find that a correction to a recorded macro is necessary, you *can* rerecord it by going through all those steps again. However, if it's just a *minor* flaw, you can simply edit the text definition of the macro that you pasted in. That may be much easier than re-recording it.

Be aware that with Keyboard Recording, **everything** you type will be recorded. That includes **Backspace** if you type something wrong and correct it. What this really means is, unless you plan on editing your keyboard definition later, if you make a mistake you will need to start over.

Keyboard recording and Power Typing

Keyboard Recording is allowed during Power Typing. However, since the status indicator is already showing **PowerType**, the usual message of **KB Recording** will not appear. However, keyboard recording proceeds as usual. When you press the key mapped to **(Record)**, which by default is the Scroll Lock key, you will see the SPFLite KeyMap dialog will appear. When you exit from KeyMap, Power Typing continues in effect. You can immediately use any keys you have added or changed in KeyMap in your resumed Power Typing session.

Saving/Editing key contents as .DO files

If you create more complex keyboard macros, and need to edit them, it is sometimes awkward to do so within the limited input box available with the KEYMAP dialog. An alternative is to store the macro in a separate file in the MACROS folder. These saved files are simple text files stored with a **.DO** file extension.

To invoke a saved .DO file, simply enter **DO name**. e.g. to invoke **ABCD.DO**, just enter **DO ABCD** in the KEYMAP entry.

DO file description

The contents of the **.DO** file are identical to the entry you would make in a KEYMAP entry field for a particular field. Except you may "spread out" the entries for multiple command over multiple lines. The **DO** process will process the lines in order.

Comments - You may insert complete line comments by starting the line with a Semi-Colon (;) These lines are totally ignored. You may also include completely blank lines for spacing and appearance.

You can also include comments on the actual data lines. This is done by leaving at least 1 blank after the real operands and starting the comment with a Semi-Colon (;

You may include **any** of the KEYMAP directives in the **.DO** file. Keyboard Primitives, Line Commands and Primary Commands.

Example:

Suppose you have a key which inserts a boilerplate log entry and it was previously defined as:

(Home)(EraseEOL)[Up Max](Enter)(FirstLineCmd)[n5](Enter)[---/G/](ISODate)[](Time)(NewLine)
(tab)

Editing could be difficult as you try and remember just what all those primitives really do. Instead this could be saved along with comments in a file named **LogEnt.DO** in your **MACRO** folder. The example shows separate comment lines, blank lines and trailing comment types.

```
; Create an empty Log entry at the top of the file
;
up max           ; Go home
(FirstLineCmd)    ; Go to top line cmd area
[n5](Enter)
; Enter some boilerplate with Date/Time
[---/G/ ](ISODate)[ ](Time)
(NewLine)
(tab)           ; Position cursor
```

Now invoke this LogEnt.DO file by entering **DO LogEnt** in the key's text box in KEYMAP

Here is a second example

The comments here explain what it is doing. The macro would be invoked by a simple **DO SUBIT** command

```
; Sample DO Macro - Copy the edit data to a new temporary tab
;
;           - Repeat the top line 5 times
;           - Replace all 11111 with 12345
;           - Replace all SSSSS with Subject:
;           - SUBMIT the resulting file
;           - Return to orig tab, leave new tab open

cut .zf .zl           ; Cut all lines
clip
; Now in the Clip tab, change the data
; Overtype the 1st line
UP max
(FirstLineCmd) (NewLine) {R5} (Enter)
change all word "11111" "12345"
change all word "SSSSS" "Subject:"
submit
; Go back to original tab
swap prev
```

Case Study: Implementing bookmarks

A number of Windows text editors have a *bookmark* capability, in which you can set several “marks” in various places in your file, and then go back and successively find them. Here is an SPFLite implementation of such a capability, using tags as the representation of bookmarks. (Not *SPFLite* bookmarks, a term used interchangeably with *label*s, but a more generalized kind of bookmark that can exist in multiple places.)

First, we arbitrarily choose a tag name to represent a bookmark; tag **:M** is selected. See [Working with Line Tags](#) for more information on using tags.

Next, document the keys and the functions they will perform, which will be based on F2. (In IBM ISPF, F2 is commonly assigned to the 3270 SPLIT-screen command, a function that is not implemented in SPFLite. So, F2 may be a good choice as an available key to use.)

We also want to keep the current cursor position when toggling bookmarks, so the toggle operation will be 'wrapped' in functions to save and restore the cursor. This example shows the use of [SaveCursor](#) and [RestoreCursor](#) functions.

- F2 = Go to next bookmark
- Shift F2 = Go to previous bookmark
- Ctrl F2 = Toggle bookmark on current line
- Alt F2 = Clear all bookmarks. In this implementation, you will see **No lines (re)tagged** if no bookmarks exist.

Finally, set the mapping of key F2 accordingly, and allow the keys to auto-repeat:

- F2 = **LOC :M NEXT**
- Shift F2 = **LOC :M PREV**
- Ctrl F2 = **(SaveCursor) { ::M} (RestoreCursor)**
- Alt F2 = **TAG :M OFF**

User Lines provide an alternative to tags for implementing bookmarks. Here is a possible implementation:

- F2 = **LOC U NEXT**
- Shift F2 = **LOC U PREV**
- Ctrl F2 = **(SaveCursor) {TU} (RestoreCursor)**
- Alt F2 = **REVERT ALL**

Case Study: Making editing easier on laptops

Many laptops have keys like *Insert* and *Delete* in unconventional and hard-to-reach places on the keyboard, and they can be tiring to use, whereas the arrow keys are usually easy to reach. SPFLite key mapping can be used to provide an alternative for these hard-to-use keys.

First, document the features we wish to make available. The functions *Insert*, *Delete* and *Backspace* are to be made available using the Ctrl-arrow keys.

Since *Delete* has the effect of “pulling” a line over to the left by one column, we also want to provide a “push” function by allowing a single blank to be inserted. It will be necessary to force the insert mode on, regardless of what it was before. So, an ordinary toggle-insert-mode command won’t work for this. We also want to retain the cursor position after inserting a blank, to mirror how deletion of one character works. To do that, we need to generate a Left-Arrow cursor movement afterwards.

Besides, if we wanted to insert characters and have the cursor ‘travel’ we could always just use an ordinary *Insert* key and then hold down the spacebar.

We also want the insert mode restored to what it used to be, and not just always turn it off, so this example shows the use of [RestoreInsert](#) instead of [ResetInsert](#).

The functions to be provided are:

- Ctrl-Left = destructive backspace one character
- Ctrl-Right = toggle insert mode
- Ctrl-Up = insert one blank character and retain cursor position
- Ctrl-Down = delete one character and retain cursor position

Finally, set the mapping of these keys, and allow the keys to auto-repeat (which is useful, since the arrow keys naturally auto-repeat):

- Ctrl-Left = **(Backspace)**
- Ctrl-Right = **(Insert)**
- Ctrl-Up = **(SetInsert) [] (Left) (RestoreInsert)**
- Ctrl-Down = **(Delete)**

Working with SPFLite

The topics in this section describe in general terms how to perform many of the functions of SPFLite.

These **Working With** topics are tutorials that provide in-depth discussions of SPFLite operation, beyond the Help descriptions you will find for individual commands.

Starting and Ending SPFLite

Contents of Article

- [Introduction: Standard SPFLite Startup](#)
- [Editing Files using Drag-and-Drop Support](#)
- [SPFLite Command Line Options](#)
- [Specifying an Overriding Profile to use](#)
- [Terminating SPFLite](#)
- [Terminating a Single File Tab](#)
- [Terminating All Tabs](#)

Introduction: Standard SPFLite Startup

Based on a General Options setting, you can choose to reopen any files that had been open the last time SPFLite was shut down.

If you enable "[Re-Open last file\(s\) at Start](#)", then when you double-click the SPFLite icon which was created during the install, you will be presented with the same file tabs that were open the last time you used SPFLite. The File Manager will be present as the leftmost tab. The tab that was active when you shut down will be presented as the active tab when you resume.

If you have not enabled "Re-Open last file(s) at Start", you will be presented with the File Manager screen containing a directory list based on your last-used File Manager display selection.

Editing Files using Drag-and-Drop Support

You can do the following with SPFLite's Drag-and-Drop support:

- Drag-and-Drop a file from Explorer or from the desktop onto the SPFLite icon to have it start editing with that file. The File Manager will be present as the first tab, and the edit file tab as the second one.
- Drag-and-Drop a file onto an existing SPFLite window. If the current tab is (Empty) the file will be opened on that tab. If the current tab contains data, or is the File Manager tab, the dropped file will be opened in a new file tab.
- A shortcut to a data file (rather than the file itself) may also be Dragged-and-Dropped onto SPFLite.

SPFLite Command Line Options

When started from the command line, operands for SPFLite normally include the initial filename to be edited. If SPFLite is started without a file name, the SPFLite startup will proceed as described above.

The following command-line options are supported. Note that all options can be specified in any abbreviated form down to a single letter. e.g. -CLIP can be specified as -CLIP, -CLI, -CL or -C.

-CLIP option

If '**-CLIP**' appears in the command line, SPFLite will operate in clipboard edit mode, which does the following:

- The contents of the Windows clipboard are opened up into a Clipboard edit session when SPFLite starts. This will directly edit the contents of the Windows clipboard, without any intermediate or temporary file involved.

- When the primary Command **END** is entered, the current edit data is returned to the Windows clipboard.

Clipboard mode is designed to provide a quick, convenient means for editing clipboard data with the SPFLite command set. From outside SPFLite, it works most effectively if a separate icon is created for SPFLite that has a command-line operand of **-CLIP**. To create a **CLIP icon** on the desktop, copy the normal icon and add **-CLIP** as an operand, in the icon's Properties window. You can now click on the icon to directly edit the clipboard contents.

If SPFLite is already running, you can use the **CLIP** primary command to open a new tab containing the current clipboard contents.

-WINE option

If '**-WINE**' appears as the command line, SPFLite will modify certain operations to assist running under WINE on *NIX systems. SPFLite **will** operate correctly under WINE but certain screen features (like TootTip Help) are problematic. **-WINE** tells SPFLite to suppress certain features which cause difficulty under WINE.

-NOLOOP option

If '**-NOLOOP**' appears as the command line, SPFLite will **NOT** activate it's normal loop detection logic. This detection logic is design to monitor for SPFLite activities which are taking far longer than expected to complete. It provides a Pop-Up message which allows the user to cancel the session, or to allow execution to continue.

However some activities can be delayed by external factors (LAN network slowdowns etc.) and the Pop-Ups can become a nuisance since there is no real underlying problem which can be corrected. **-NOLOOP** simply requests SPFLite to forgo monitoring for such loop conditions.

-BROWSE option

BROWSE mode allows you to use SPFLite to view data files in read-only mode without risk of changing the data. This is particularly important if you normally open files with the Profile set to **AUTOSAVE ON**, or if you have opened an important file and want to be sure it does not get modified.

In **BROWSE** mode, you are prevented from using any action (Primary Command, Line Command or simple typing) which would modify the file's data.

-BROWSE may be coded alone as an operand, in which case a [BROWSE](#) dialog will be presented where you select a file to browse. Or, it can be coded as **-BROWSE filename**, to start SPFLite with the specified filename opened for browsing.

To create a **BROWSE icon**, copy the normal icon and add **-BROWSE** as an operand, in the icon's Properties window. You can now Drag-and-Drop files onto the icon to quickly Browse them.

-VIEW option

VIEW mode allows you to use SPFLite to view data files in read-only mode without risk of changing the data. This is particularly important if you normally open files with the Profile set to **AUTOSAVE ON**, or if you have opened an important file and want to be sure it does not get modified. Unlike **BROWSE**, **VIEW** allows you to make changes to the file, they will simply not be saved. If you make changes to a Viewed file, the word **View** in the left-hand Status Bar box will be displayed as **View** to remind you.

In **VIEW** mode, the **SAVE** and **REPLACE** commands are disabled, as is the **AUTOSAVE** function of the **END** command. The **CREATE** command is allowed in **VIEW**, so you can create *other* files without changing the one you are browsing. You may also save the entire file you are browsing under a new name by using **SAVEAS**. (If you happen to use the **SAVEALL** command from any open file, it will only save files opened for

edit, while files opened for browse are ignored.)

-VIEW may be coded alone as an operand, in which case a **VIEW** dialog will be presented where you select a file to View. Or, it can be coded as **-VIEW filename**, to start SPFLite with the specified filename opened for viewing.

To create a **VIEW icon**, copy the normal icon and add **-VIEW** as an operand, in the icon's Properties window. You can now Drag-and-Drop files onto the icon to quickly View them.

-INIT option

The **-INIT** option causes SPFLite use a different **INI** file for customization, instead of **SPFLite.INI**. This allows you to have INI files with different global options such as handling of file deletions, fonts and screen colors, mouse and keyboard options, etc. (Note that keyboard *mappings* are *held* in the **SPFLite.KBD** file, which cannot be overridden like the INI file can.)

To use, add **-INIT initname** to the command line, where **initname** is either:

- the **base name** of the alternate INI file. For example, to start SPFLite using an INI file named **CUSTOM.INI** the command line would be **-INIT CUSTOM**. The INI file will be searched for in the normal SPFLite data path (Documents\SPFLite).
- A **fully qualified path** to the INI file.

Example: **-INIT "D:\Data\Editors\SPFLite\CUSTOM.INI"**

-KEYMAP option

The **-KEYMAP** option causes SPFLite to launch the KEYMAP dialog immediately after startup, and prior to beginning normal editing operations. You might wish to use the **-KEYMAP** option in case you had a serious KEYMAP configuration issue that was preventing you from operating SPFLite properly, such as the absence of a properly defined ENTER key.

Because use of the **-KEYMAP** option is a type of "emergency startup" contingency, SPFLite will not have completely finished all of its initialization steps when you get the KEYMAP dialog. As a result, the dialog will be displayed with a default font rather than your customary editing font. After you finish, and then SPFLite begins normal operation, the customary editing font will appear when you issue a KEYMAP command again.

Example: The command line:

SPFLITE .EXE -B -I CUSTOM MyTestFile .txt

Requests SPFLite to start in Browse using the CUSTOM.INI file for customization, and to Open MyTestFile.txt as the initial working file.

Specifying an Over-riding Profile to use

When you specify a filename on the command-line to be opened (including the **-BROWSE** and **-VIEW** variations) it is possible to specify a different Profile name to be used to process the file. This is the same ability that is provided by the Primary **EDIT**, **BROWSE** and **VIEW** commands.

Simply provide the overriding Profile name, preceded by a . (period) on the command line following the filename. If the filename was enclosed in quotes, the Profile name follows this trailing quote, and is not itself quoted.

e.g. SPFLite -B "My File To Browse .txt" .NewProfile

The above shows a request to browse a file using the special profile called 'NewProfile'.

Terminating SPFLite

When SPFLite is terminated, it will 'remember' the current screen position and size and will use these values the next time it is started. If SPFLite was Maximized at termination time, it will be restarted in Maximized mode.

As with startup, the other actions performed at termination are affected by your Option choices. There are two levels of termination, the first is the termination of a single file tab, the second is termination of all file tabs and SPFLite shutdown.

Terminating a Single File Tab

A single file tab is terminated with the **END** command. This can also be accomplished by right-clicking on the desired tab. When you use a right-click in this way, SPFLite treats it as identical to the **END** command, or to a key mapped to the **END** command.

- If the tab is in View mode, and the file has not been modified, it is simply closed. If you have modified the file during View, then the action depends on your choice for the Options -> General selection of [Warn on modified View file](#). If you chose to be warned, then you will be given an option to return to View or just exit; If you chose *not* to be warned, the file is closed without saving.
- If the tab is in CLIP mode, the current text contents are returned to the Windows clipboard and the tab is closed.
- If the tab is in EDIT mode, the contents will be either saved or not saved, based on the Profile **AUTOSAVE** option for the file's file type; and then the tab will be closed.

For all above cases, if there are still remaining active tabs, control will be passed to the adjacent tab.

If there are no further active tabs, what happens next is based on whether you enabled "[Close File Mgr with last tab](#)" in the File Manager Options. If you enabled this, SPFLite will terminate. Otherwise, control is passed to File Manager.

When the **END** command is issued in the File Manager, it does not 'close' it, but causes the file display to move up to the next higher directory level, or back from a File List display to a directory display. Repeated use of the **END** command will eventually leave the File Manager displaying the root directory of the selected drive, which for most users will be the `C:\` directory, and at that point **END** will have no further effect.

Terminating All Tabs

If the Windows standard close button **[X]** in the title bar is clicked, or the **EXIT** command or the **=X** command is issued, all file tabs will be processed as if an **END** command had been entered and SPFLite itself will terminate.

There are some differences between using the close button **[X]** to close SPFLite, and using **END** on each tab first.

Just so we're clear here, don't confuse "**the [X] button**" with the "**=X command**". The **[X] button** is a Windows icon that looks like an **X** on the right side of the Title Bar of the SPFLite window that you click on with the mouse, whereas the **=X command** is an SPFLite command that is identical to the **EXIT** command and is typed on the primary command line by you as the characters **=X** followed by the ENTER key.

When you use the close button **[X]**, SPFLite first remembers all currently active files, so that when you restart SPFLite later, all these files will be reopened, and then all tabs, including File Manager, are closed.

If you use **END** individually for every file, they are all now "closed" and so when you restart, there are no previously open sessions to be resumed, and so none of them will be automatically reopened.

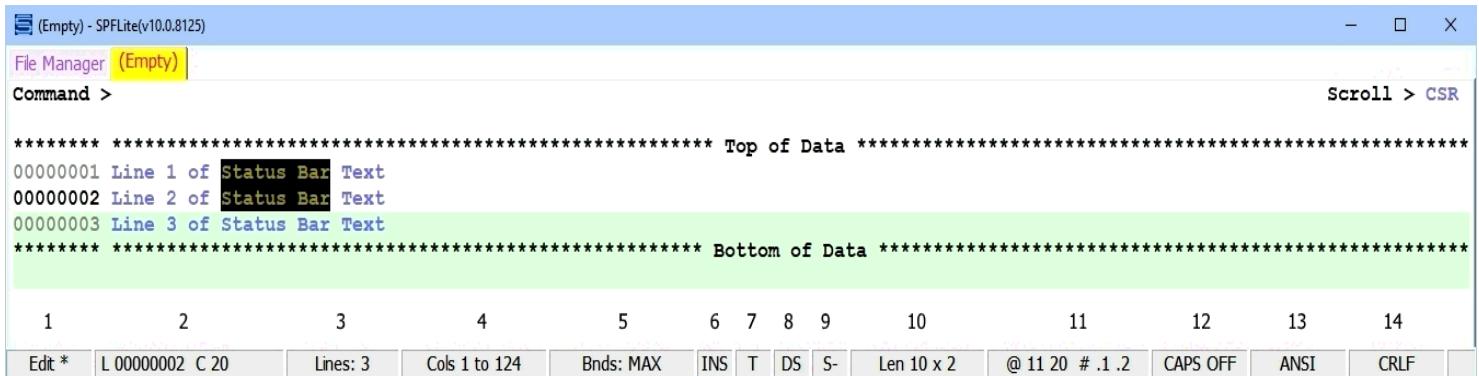
The **EXIT** command can also take an optional **NOREOPEN** operand. If **NOREOPEN** is specified, SPFLite will not save the list of currently open Edit tabs for use at the next normal SPFLite start. This means the next SPFLite startup will begin in the File Manager tab, with no other tabs being automatically loaded.

Whether SPFLite re-opens files at start-up is controlled by a preferences selection on the [Options -> General](#) screen. The **NOREOPEN** option will cause the list of currently-open files to be discarded, even if the Re-open checkbox is enabled. (The files themselves are not discarded - only the list that refers to them is discarded.)

When your options are set to re-open the previous set of files at startup, these files will only be opened by the **first** instance of SPFLite that starts. If your settings allow multiple instances of SPFLite, then 2nd and subsequent instances will **not** attempt to re-open the set of files again.

Status Bar Contents

The bottom line of the SPFLite window is reserved for the Status Bar. This contains a variety of information regarding the current file being worked on and is constantly updated. The status line looks like this:



The above display shows a status bar with all possible boxes displayed. You can choose yourself which boxes to display and in what order, from left to right you wish them to appear. See "[Options - SBar](#)" for how to alter this.

The boxes, and their Names and Descriptions follow:

Box 1 (MODE) This will show either **Edit**, **Browse**, **View**, **Clip** or **Set Edit**. Edit is the standard operating mode. Browse can be started with a **B** line command in File Manager, or by using a **BROWSE** command. View with a **V** line command or **VIEW** command. Clip is displayed when you are directly editing the Windows clipboard, and is entered by using the **CLIP** command. If you are editing the list of **SET** variable values, you will see Set Edit. Some of these various modes can be started using various command-line options. See [Starting and Ending SPFLite](#) for details.

If you open a Read Only file, whether via Edit or Browse, SPFLite will open it in Browse mode, and then change the operating mode indicator to **RdOnly**. The **RdOnly** indicator is a reminder that the file has the Read Only attribute, but otherwise is identical to being in Browse mode. See [Working with Read Only Files](#) for more information.

When the data in the tab has been modified and not yet saved, an ***** will appear following the descriptive text. For example, modified edit file will show **Edit ***

When the tab is in Multi-Edit mode, the word **Edit** will be preceded by the number of files currently loaded in the tab and the modified indicator (*****) will be preceded by a number indicating how many files have been modified. If 3 files are loaded, and 1 is modified, the box will contain **3 Edit 1***

Note: The ***** modified indicator will appear right away, as soon as you type something to modify the file. You will also see the name of the file change color in the file tab for it on the top of the screen, but the change in file-tab color will happen only when you press Enter or issue a primary or line command (either directly or via a mapped key). So until you do one of those things, the ***** modified indicator and the presence of the file tab's modified color may be briefly out of sync. It's just a design limitation about how quickly the screen can be updated, and is nothing to worry about.

Box 2 (**LINNO**) If the cursor is located within the text area, this area shows the line number as L 000123 then column number as C 1, and then an optional tag name when a line has both a label and a tag at the same time. When a command like **FIND ABC X DX** causes the cursor to be in the "interior" of an excluded region, the Line/Column display will appear in reverse video. When there is no meaningful column number (like when entering a line command) it will not appear here. When the cursor is not on a data line (like on the primary command area) the entire Line/Column display will be blank.

If the cursor is logically located on an excluded line, the box will display in white on green to highlight this condition. So, if the cursor is on column 8 of excluded line 4, you will see **L 00004 C 8**

If the cursor is not within a data area or line command area, then this field will show the Last Modified Date/Time of the edit file. This will be displayed as standard ISO date and time format. e.g. yyyy-mm-dd hh:mm:ss

Box 3 (**LINES**) The current number of text lines in the file.

Box 4 (**COLS**) The current visible range of text columns.

Box 5 (**BNDS**) The current BOUNDS setting. If no BOUNDS are set it will say **BNDS: MAX**, if Bounds are active it will display the left and right column boundaries. When the **BOUNDS** setting is anything other than **MAX**, the status line display will show the **BOUNDS** setting in white letters on a red background, like **Bnds: 1 to 40** so that it can't be ignored. This will help users to avoid the unexpected and nonstandard handling of data that occurs when non-default bounds are in effect, if that was not their intent.

Box 6 (**INSOVR**) The current Insert mode. INS = Insert mode OVR = Overtype mode. If you have invoked the ([DataInsert](#)) primitive to enter Data Insert mode, it will show as **INS**.

Box 7 (**CASE**) The current [CASE](#) setting. **C** = Case sensitive **T** = Case insensitive. The **C** and **T** codes have the same meaning as **C** strings and **T** strings on **FIND** and **CHANGE** commands.

In addition, the 'search context' used as a default by **FIND/CHANGE** commands is shown here. When the search context is **CHARS** mode, the box will contain **C** or **T**. When the search context is **WORD** mode, the box will contain **C W** or **T W**. The search context is altered by the [FIND](#) command, and by the [Use WORD as the default for FIND/CHANGE](#) option of "[Options - General](#)".

Box 8 (**CHANGE**) The current CHANGE column/data shift mode. DS will be displayed for normal Data Shift mode, CS for Column Shift mode. See [Change Data Shift Modes](#) for details. Because Data Shift is the default shifting mode, as traditionally supported by ISPF, you can also think of DS as meaning Default Shift.

Box 9 (**STATE**) The current status of the associated STATE file for this edit. If **S+** is displayed, it indicates that there **is** a currently existing STATE file. If **S-** is shown, no existing file exists. See [Saving the Edit STATE](#) for more information.

Box 10 (**MISC**) Used for various important information messages, such as when Power Typing or [recording](#) is active.

When there are no messages being displayed here, one of these information fields will be present:

- If no data is being highlighted, the length of the current line is shown as **Line Len 0123**

- If a single character is highlighted, its hex and decimal value are shown, such as **X'32' = 50** for the digit '2'
- If multiple characters are highlighted on a single line like **ABCDE**, you will see **Len 5**
- If a block 5 characters long by 3 lines were highlighted, you will see **Len 5 x 3**

For larger files, there is a small “underline display” at the bottom edge of this status box. You will see it slide left and right as you scroll up and down in the file. It is intended to give an approximate “gauge” of where you are in the file, somewhat like Windows does with a vertical scroll bar.

Box 11
(SELECT)

The current text selection values. This can be:

Blank	There is no current selected text area.
@ nn mm # .L1 .L2	If there is a currently selected area the two numbers following the @ are the starting and ending columns of the selected text. The values after the # are the starting and ending line range. The @ and # characters are reminders that these are the substitution characters that can be entered as command line operands to substitute in the displayed values.
Select	If the word 'Select' is displayed, it indicates that values exist for a selected area, but they are currently inactive. The status of a selected area can be toggled between active and inactive by either left-clicking on this box, or by using a key assigned to the (ToggleSelect) keyboard primitive.

Box 12 (**CAPS**) The current [CAPS](#) mode. This field can contain one of four values:

CAPS ON
CAPS OFF
CAPS AUTO: on
CAPS AUTO: off

The last two bear explanation.

In ISPF, no matter what you set your **CAPS** mode to, the ISPF editor will examine your file when you open it. If it finds data which is contrary to your **CAPS** setting, it will forcibly change the **CAPS** mode **ON** or **OFF** to conform to whatever is in your data, and that **CAPS** setting will be permanent (until you change it, or ISPF changes it again in the same way). Experience has shown that the way that ISPF handles this can be problematic.

In SPFLite, when you set **CAPS** mode to **ON** or **OFF** for a given file type, it stays that way **until you alone change it**. SPFLite **never** changes the **CAPS** mode from **ON** to **OFF** or vice versa, by itself.

Note that when **CAPS ON** is in effect, while you are typing in new characters on a data line, they will appear in the "Text - High-Intensity" font color. This is done as a visual reminder that the alphabetic data you are entering is being forced to upper case (this is regardless of the Caps Lock key). If that is not what you wanted, issue the **CAPS OFF** command. Using **CAPS ON** is common for mainframe data, while **CAPS OFF** is more typical of PC data.

If you set **CAPS** mode to **AUTO**, the caps mode is “conformant”. The file is examined, comparable to how ISPF does it, but in SPFLite this setting is only temporary. When you open the file again, the file examination is repeated. The “on” or “off” displayed in *lower case* is a reminder that the current automatic/conformant caps mode is **temporary**, and is **not** stored in the PROFILE, whereas **CAPS ON** and **CAPS OFF** is stored in the PROFILE.

The **AUTO** part of AUTO: on or AUTO: off is stored in the PROFILE, but the **on/off** setting itself is **not** stored.

- Box 13 **(SOURCE)** The current character set being used. This is controlled by the [SOURCE](#) Profile setting. The default is ANSI. Other possible values are EBCDIC and various Unicode settings.
- Box 14 **(EOL)** The current [EOL](#) setting. This shows how the End Of Line value has been set in the PROFILE. It defines how SPFLite determines where the end of a line is located. The standard ending for Windows text files is CRLF. **EOL** may display **CRLF**, **CR**, **LF**, **NL**, **AUTO**, **AUTONL**, or two or four hex digits. It may also be set to **NONE**, meaning there are no terminators. **NONE** requires an explicit record length to be defined with an **LRECL** profile option.

When you have EOL AUTO or EOL AUTONL in effect, and PAGE ON is also in effect, this field will display the current page number in the edit screen, in the format of **Pg: 1 of 5**. Because this replaces the AUTO or AUTONL state of EOL, you would have to issue a PROFILE command to determine which of these specific EOL settings were in effect. Users of SYSOUT files normally do not change the AUTO/AUTONL setting once it's defined, so for most people, omitting the AUTO/AUTONL display here should not present an inconvenience.

Basic Edit Functions

Contents of Article

- [Introduction](#)
- [Direct Modification of Text](#)
- [Primary Commands](#)
- [Visual indication of a file's modified status](#)
- [Command Macros](#)
- [Line Commands](#)
- [Edit Commands and Command Key Processing](#)

Introduction

There are a variety of basic interactions available to alter your edit file.

Direct Modification of Text

You may modify text by using the arrow keys to move within the data, and type over or insert text as desired. The Insert key will toggle Insert / Overtake mode as needed. (The current status is always visible as INS or OVR in the [Status Bar](#)) You can control the size of the cursor in normal mode and in Insert mode and whether it blinks. See [Options - Screen](#) for customization options. If automatic colorization support is active, the display line is not re-colored to match the entered text until the next "attention" key is pressed (Enter, command key, etc.)

If the Keyboard Option "KB Starts in INSERT mode" is checked, the keyboard will be in INSERT mode when SPFLite is started. Note that editing in Insert mode is typical for Windows-based editors, while editing in Overwrite mode is how the mainframe 3270-based ISPF operates. Even with the many word-processing enhancements available in SPFLite, most users will find it more practical to operate SPFLite in Overwrite mode most of the time, rather than starting off in Insert mode.

You may sometimes see "Overwrite mode" described as "Overtake mode" in the documentation. They both mean the same thing.

If HEX mode is selected (see the [HEX](#) command), then data may also be edited in Hex, which can be used to enter characters which are not normally available via the keyboard. Note that if you use Hex mode to insert/alter control characters such as CR and LF, you must do so carefully. Hex mode is sensitive to the file's **SOURCE** encoding. If editing an EBCDIC file, for example, the hex characters entered must be the EBCDIC values, such as X'F1' for EBCDIC '1' rather than X'31' for ANSI '1'.

If you want to enter special characters but don't want to use HEX mode to enter them, you can place them into the clipboard using the (CharSet) popup and then paste them into your file. See the (CharSet) keyboard function in [Keyboard Primitives](#) more information.

Primary Commands

Primary commands are entered on the Command line at the top of the screen. They typically affect multiple lines in the file being edited. Multiple commands may be entered at one time on the command line, separated by a command separator character. As in ISPF, the default for this character is the ; semicolon, but this can be altered in the [Options - General](#) options window. When the character assigned as the command separator is enclosed in quotes, it is treated as ordinary data.

If the command line is prefixed with an & character, then the command line will be retained in place following command execution rather than being cleared. This allows you to perform the command again by just pressing Enter.

Primary commands are used for many purposes:

- Scroll to a specific line number or [Line Label](#).
- Find a specific line that contains (or does not contain) a search string
- Find and change a character string
- Save the edited data or cancel without saving
- Create or replace other files than the one being edited
- Sort data
- Delete lines
- Undo changes made, subject to the number of maintained Undo levels (see [SETUNDO](#) for more details)
- Initiate parallel editing of lines in Power Typing Mode
- Submit jobs for execution using an external process
- Execute external programs using **CMD**

You can now use the [LINE](#) primary command to apply line-mode or block-mode line commands to a range of lines.

Visual indication of a file's modified status

When a file has been modified since it was initially loaded, or since the last **SAVE** command, this will be indicated on the file tab by the color of the file name itself. See [Options - Screen](#) for choosing these colors. You can choose colors that will best convey the status of each file. For example, you could choose to display modified files with red letters for the file tab, and blue letters for unmodified files, with a light background for each. Then, when you see a file tab in red, you would know the file is modified.

Note: It is **up to you** to select colors that will work for you to achieve this affect. At installation time, the default color scheme will **not** do that, but will require user modification. You may need to experiment to find good color choices.

There is a second file modification indicator - the word **Edit** in the lower-left corner of the screen on the status line. In unmodified files, you will just see **Edit**, while in modified files, you will see **Edit ***.

In Multi-Edit mode, the modification indicator display is more complex. See [Working with Multi-Edit Sessions](#) for more information.

Command Macros

If a command entered on the Command line is not recognized by SPFLite as one of the built-in commands, before deciding that a command is invalid, SPFLite will first check to see if the entered command is a user-defined command macro. A command macro is script file with an extension of .MACRO stored in the **\SPFLite\MACROS** data folder. If such a file (e.g. cmdname.MACRO) exists, then SPFLite will execute the script file. Further details on using command macros is provided in [Command Macro Support](#).

Because of this search order, a user-defined command macro cannot have the same name as a built-in primary command. You *could* save a command macro file with a primary command name, like **FIND .MACRO**, but SPFLite would never use it. (This was a design trade-off, for performance reasons).

Command macros should not be confused with **keyboard macros**, which is a completely different feature. See [Keyboard Customization and Keyboard Macros](#) for more information.

Line Commands

Line commands affect single lines or block of lines. You enter line commands by typing them in the Line Command field on one or more lines and then pressing Enter. The line command field is represented by a column of 6-digit numbers on the far left side of your display. When you are entering data into new temporary blank lines created by the **I** line command, the line command field contains 6 quotes. This field is also used to

display special lines, such as the ==CHG> flag, which indicates a line which has been altered by a primary **CHANGE** command. You can use line commands to:

- Insert or delete lines
- Repeat lines
- Rearrange lines or overlay portions of lines
- Split lines apart and join lines together
- Perform text paragraph entry and formatting
- Shift data
- Include or exclude lines from the display
- Control tabs and boundaries for editing
- Alter the text case (upper-case, lower-case, sentence-case and title-case)

Edit Commands and Command Key Processing

When text is entered on the command line, and a command key is pressed rather than the ENTER key, SPFLite concatenates the contents of the command line to the definition of the command key. The result is handled as a single, composite command by SPFLite. (A "command key" is a key mapped to a primary command.)

For comparison purposes, in IBM ISPF, a 3270 Enter key or PF key would be treated as a "command key", while data keys, cursor keys and PA keys would not be. 3270 data and cursor keys do not initiate a 3270 data transmission, but the Enter and PF keys do. A PA key like PA1 only transmits the fact that the PA key itself was pressed, but doesn't send any other data; it's like an "interrupt" key. "Command keys" in SPFLite use a similar concept to the 3270 Enter and PF keys.

For example, when you use a Command key defined as a scroll command (**UP**, **DOWN**, **LEFT**, or **RIGHT**), the value entered on the command line becomes the **operand** of the scroll command. Assume you have the **PageDn** key mapped to the **DOWN** primary command. Entering **H** or **HALF** on the command line and then pressing the **PageDn** key results in a composite command of **DOWN HALF** being issued to scroll the screen down by half its height.

You may choose to override this concatenation when you set up the command key definition by preceding the command with a ! (exclamation mark character). This causes the programmed command to **replace** whatever is in the command line. e.g. If you have a key programmed for **RESET**, it would probably be wise to prefix it with a ! to prevent the **RESET** command processor being confused by possible command line 'remnants' existing when the command key is pressed.

Edit Boundaries

Boundary settings control what data is affected by other line and primary commands. You can change the boundary settings by using either the [BNDS](#) line command, or [BOUNDS](#) primary command.

The Bounds column range is in effect unless you specify overriding boundaries when entering a command. The action of some, but not all, primary commands are modified by changes to the BOUNDS setting. Refer to the individual command descriptions for the effect the current bounds settings have.

If you do not explicitly set bounds, the editor uses the default bounds, which are the entire data line. When the bounds are the entire line, the editor may be said to be operating in "unbounded mode", which will cause a status line display of **Bnds: MAX** to appear. You can set the editor to operate in unbounded mode by issuing a primary command of **BOUNDS 1 MAX**, and typically this is how SPFLite is operated.

Note: When the **BOUNDS** setting is anything other than **MAX**, the status line display will show the **BOUNDS** setting in white letters on a red background, like **Bnds: 1 to 40** so that it can't be ignored. This will help users to avoid the unexpected and nonstandard handling of data that occurs when non-default bounds are in effect, if that was not their intent.

See the [BNDS](#) line command and the [BOUNDS](#) primary command for further information.

A word about the use of BOUNDS

The BOUNDS feature of ISPF is one that IBM did not extensively document, and in practice, mainframe ISPF users do not tend to use this feature very often. Every effort was made to implement BOUNDS in SPFLite in an ISPF-compliant manner. However, it may produce surprising and unexpected results if you are not familiar with the actions taken by various commands when operating under restricted column BOUNDS. The "surprising and unexpected" aspect is even more of a factor for SPFLite users without a prior mainframe ISPF background.

For many users, you will likely get the most benefit from SPFLite by operating in unbounded mode most or all of the time, and not worry about using BOUNDS unless you have very particular editing requirements.

Scrolling the Text

Contents of Article

Introduction

Primary Commands UP, DOWN, LEFT, RIGHT and LOCATE

Arrow Keys: Up, Down, Left and Right

Primary Commands TOP and BOTTOM

Mouse Wheel Scrolling

Primary Command LOCATE

Primary Commands FIND and NFIND

Introduction

There are a variety of methods available for scrolling the edit window to the area of text you wish to work on.

Primary Commands UP, DOWN, LEFT, RIGHT and LOCATE

The first four of these commands are typically mapped to keyboard keys for easy availability. Their full syntax is available in the Primary command section ([UP](#), [DOWN](#), [LEFT](#), [RIGHT](#)). **UP** and **DOWN** are typically mapped to F7 and F8 and/or PageUp and PageDn. **LEFT** and **RIGHT** are typically mapped to F10 and F11.

Although the amount to be scrolled can be provided as an operand to these commands (other than Top / Bottom), the default is visible and modifiable in the **Scroll >** field in the upper right of the screen, such as "HALF" shown here:



The Scroll amount can be changed by simply typing over it. You need only type the first letter of one of the symbolic scroll amounts listed below, like **P**, **F**, **H**, **D** or **C**; when you press Enter, the full name of the symbolic name will appear. For example, if you had **HALF** displayed, you could just type over the **H** with a **P**, and momentarily it would appear as **PALF**. When you pressed Enter, **PALF** will be changed into **PAGE**. The default Scroll amount will be used whenever one of the scroll commands is issued without an operand. The Scroll amount is saved as part of the Profile settings for each file type.

You may set the scroll amount to one of the following symbolic scroll amounts:

PAGE Scroll by the number of text lines on the screen (vertical) or the number of columns (horizontal). If the current file Profile is set to **EOL AUTO** or **EOL AUTONL**, then the scroll will be to the relative =PAGE> line of the previous/next page.

FULL Handled the same as **PAGE** but will always ignore **EOL AUTO** and **EOL AUTONL** status.

HALF Scroll by half the number of text lines on the screen when scrolling vertically, or half the number of columns when scrolling horizontally

DATA Scroll by the number of text lines on the screen minus one when scrolling vertically, or number of columns minus one when scrolling horizontally

CSR	Cursor scrolling. Move the line /or column containing the cursor to the edge of the screen, based on the scrolling direction.
nnnn	Scroll by a fixed number of lines or columns. nnnn may be 1 to 4 digits.

Arrow Keys: Up, Down, Left and Right

By default, when the cursor reached the edge of the screen, it will wrap back to the opposite edge. Under [Options - Keyboard](#) you may activate keyboard scrolling; instead of wrapping as described above, the screen will scroll one character or line at a time to keep the cursor in the active edit area.

Primary Commands TOP and BOTTOM

These two commands will move the visible window to the Top or Bottom of the text respectively. **BOTTOM** may be abbreviated as **BOT**.

Most users will have the PageUp key mapped to **UP**, and the PageDown key mapped to **DOWN**. If you put an **M** on the command line and press PageDown, the same action will take place as is performed by the **BOTTOM** command. The **M** option is short for **MAX**, and **BOTTOM** is a synonym for **DOWN MAX**.

Likewise, if you put an **M** on the command line and press PageUp, the same action will take place as is performed by the **TOP** command. **TOP** is a synonym for **UP MAX**.

Mouse Wheel Scrolling

If you activate this option by setting Auto-Scroll > 0 ([Options - General](#)) then the mouse wheel can be used to scroll the text vertically, or horizontally if holding the Shift key at the same time. The number of characters or lines to scroll is specified in the [Options - General](#) setting. If you hold the Ctrl key down while scrolling with the mouse wheel, scrolling is done 4 times faster; this is called Turbo Mode scrolling.

Primary Command LOCATE

The primary command Locate (**LOCATE**, **LOC** or **L**) can be used to quickly move the top line of the screen to another position. It supports a variety of positioning modes, from moving to a specific line number or label, or to a specific Page if in [PAGE](#) mode, to generic searches by type of text line. Review [LOCATE](#) for specific details. The **LOCATE** command in SPFLite is much more powerful and flexible than in IBM ISPF.

LOCATE can also be used for the side-effect of excluding or unexcluding a range of lines, without locating any particular line.

Primary Commands FIND and NFIND

The [FIND](#) and [NFIND](#) commands may be used to move the screen to a specified location based on the contents of the text data.

Finding and Changing Data

Contents of Article

[Introduction](#)
[Specifying the Search String](#)
[Simple string](#)
[Delimited String](#)
[Picture String and Format String](#)
[Regular Expression String](#)
[Mapping Expression String](#)
[Macro Controlled String Change](#)
[Hexadecimal String](#)
[Character and Text Strings](#)
[Effect of CHANGE Command on Column-Dependent Data](#)
[Starting Point and Direction of the Search](#)
[Scroll Alignment using TOP](#)
[Qualifying the Search String Context](#)
[Truncation following the CHANGE](#)
[Delimiters used to determine Word, Prefix and Suffix boundaries](#)
[Limiting the search to specific columns](#)
[Limiting the search to Excluded or Non-Excluded lines](#)
[Limiting the search to User or non-User lines](#)
[Limiting the search to specific highlighted strings](#)
[Repeating the FIND and CHANGE commands](#)
[Case-Conformant Change Strings](#)
[Changing the Default Search Context](#)
[Restrictions on the use of LAST and PREV with Regular Expressions](#)

Introduction

FIND, and **CHANGE**, (and the negative version **NFIND**) allow you to find a specified search string, and /or change one search string to another based on a specified search string. These commands provide powerful editing functions because they operate on a complete file rather than on a single line.

The characteristics of each command follow:

FIND	Searches for the specified string and moves the cursor (scrolling if necessary) to the first occurrence of the search string.
CHANGE	Causes the same effect as FIND , but it also has a second string operand (string-2). During a search, whenever string-1 is found, the editor replaces that string with string-2. Data to the right is shifted, if necessary.
NFIND	Is like the FIND command except it searches for text lines which do not contain the specified search string. Because the search string itself is not present , NFIND only looks for <i>lines</i> which do not have the string, and only whole lines are found.

Specifying the Search String

The primary control for any search is the search string, because it represents the value for which you are looking. Two operands, string-1 and string-2, are required for the **CHANGE** command to specify the replacement value of the string once the search string is found. The rules for specifying string-1 and string-2 are mostly the same,

except that if you type a single asterisk for either one, the previous value from the previous **FIND** or **CHANGE** command is used again.

SPFLite allows you to specify the following kinds of strings:

Simple string

Any series of characters not starting or ending with a single quote ('), accent quote (`) or double quote (") and not containing any embedded blanks. A string which is also a command keyword, like **ALL**, cannot be specified as a simple string but must be quoted.

Note: Refer to the syntax description of each command for the complete list of reserved keywords for each command. SPFLite defines a number of extended keywords not used in ISPF, so if you are coming from a mainframe background, this could be important to you.

Delimited String

Any string enclosed (delimited) by either single quotes ('), accent quotes (`) or double quotes ("). The beginning and ending delimiters must be the same character.

Note: **SPFLite does not use quote-doubling** to represent quotes as data, as some programming languages do.

To use quotes as data, use one type of quote as the delimiters, and another type as the "quoted quote". Example:

<code>"It's correct"</code>	to quote a quote this way; this is standard IBM ISPF usage
<code>`It's also correct`</code>	to use accent quotes in SPFLite
<code>'It isn't correct'</code>	to use quote-doubling like this is wrong

Picture String and Format String

Any quoted string of characters, preceded by the character **P** or **F**, such as `P'>>>###'`. The picture / format strings provide a powerful pattern matching ability to enable searching for data of certain types, rather than by specific actual characters. A Picture string can be the Find string or the Change string, but a Format can only be a Change string. A Format string is a Picture-like string that operates under slightly different rules than a Picture string does. The creation of Picture / Format strings is fully covered in [Specifying A Picture or Format String](#).

When a **FIND** or **CHANGE** search picture uses the special characters = or . and a character that cannot be displayed is found, that character's hexadecimal representation is used in the confirmation message that appears in the upper-right corner of the Edit or Browse screen. For example, the command `FIND P'..'` could result in the message `CHARS X'0415' found`.

Regular Expression String

Any quoted string of characters, preceded by the character **R**, such as `R'abc$'`. The regular expression string provides an industry standard syntax for specifying search strings. The creation of Regular Expressions is fully covered in [Specifying A Regular Expression](#).

Macro Controlled String Change

There are times when the format of the 'To' string is not a constant, nor a simple re-arrangement of the 'From' string data. e.g. it requires some form of logical reasoning to determine what the 'To' string should be. A *Macro Controlled String Change* is one which passes control to an SPFLite MACRO to determine what

exactly the CHANGE should perform.

This is done by coding an **E** type literal, where the contents of the literal are the name (and possible operands) of an SPFLite MACRO. e.g. **E"MyMacroName aaa bbb"** which will cause the macro MyMacroName to be invoked with operands of aaa bbb. The macro is able to examine the found string and perform any needed logic of it's own before returning a value to CHANGE to use as the 'To' string.

This is more completely described in "[Writing a MACRO for a macro controlled string CHANGE](#)"

Hexadecimal String

Any quoted string of Hex characters (0123456789ABCDEF), preceded by the character X, such as X'41CF'. The string must be an even number of valid hex characters. Note that when you look for Hex values, it is dependent on the encoding of the file. In ANSI, the digit 1 is X'31' while in EBCDIC it is X'F1'. Hex digits greater than 9 can be in upper or lower case.

If you are looking for a known hex value, then hex strings are what you need. If you are looking for some unusual hex value in your data - but you don't know exactly what it is - there are a number of things you can do:

- You can put the editor in HEX mode, and view the data that way. If you can find the data somewhere, you will see the hex value for it.
- If you can see the character (assuming it's displayable), you can bring up the ANSI popup window, and find it there. Then, just look at the 'edges' of the ANSI window that show the encoding, and add-up the row and column values to determine the hex value for it.
- If the character is considered "undisplayable" in "Picture" terms, you can try finding it with the **P' . '** notation. Be aware that the roster of characters considered to be undisplayable might need to be configured or adjusted to your needs. See [Options - General](#) for more information. Be aware that the **P' . '** notation finds characters that are **not** in the "Normal" characters list in the General Options dialog, which may be opposite of what you expected - so be careful not to get confused by that.

For example, an ANSI encoded file, **FIND X'3132'** would be equivalent to **FIND '12'**

Character and Text Strings

Any quoted string of characters, preceded by the character **C**, such as C'conditions for' or **T**, such as T'some text'.

When using the normal simple or delimited strings (see above) , SPFLite uses your choice for the [CASE](#) setting to determine whether simple literals are to be treated as case-sensitive or case-insensitive when performing comparisons. Explicitly specifying the **C** or **T** literal type will override the [CASE](#) default for this individual command.

The Character string literal is used to direct that a proper case sensitive comparison is to be made.

For example, this command:

```
find ALL 'Condition No. 1'
```

would (assuming CASE = T) find all of the following:

```
CONDITION NO. 1
Condition No. 1
condition no. 1
coNDITION nO. 1
```

however, the command:

```
find ALL C'Condition No. 1'
```

would find only:

```
Condition No. 1
```

Note: You must use quotes if a string contains embedded blanks or commas, or if a string is the same as a command or keyword. You delimit strings with quotes, either ' single-quotes, ` accent-quotes, or " double-quotes.

For example, if you want to change the next occurrence of **every one** to **all**, type:

```
CHANGE 'every one' 'all'
```

If you left off the quotes and did this:

```
CHANGE every one all
```

the **all** would be taken as the command keyword **ALL**, and SPFLite would try to change **all** occurrences of **every** into **one**, which is not what you had in mind.

Effect of CHANGE Command on Column-Dependent Data

Historically, SPFLite has mimicked the processing of ISPF as to how the **CHANGE** command handles data which is column oriented. This method of processing is data dependent as it varies depending of the presence or absence of columns within the data.

This original processing is referred to as Data Shift mode and is still the default mode in SPFLite.

Data Shift Mode (DS)

Column-dependent data is groups of non-blank source data separated by two or more blanks, such as a table, or source code with comments starting half-way across the line. When you use [CHANGE](#) to change column-dependent data, the Editor attempts to maintain positional relationships. For instance, if you change a long word to a short word, the editor pads the short word with blanks. This padding maintains the column position of any data to the right of the change by preventing it from shifting left.

When only one blank separates words, as in most text data, padding does not occur. Changing a long word to a short word causes data to the right of the change to shift left.

Because Data Shifting is the default behavior, and is the way in which ISPF has always operated, you can think of **DS** as meaning either Data Shift or Default Shift, whichever you find easier to remember.

Column Shift Mode (CS)

SPFLite has introduced the option of Column Shift mode to the [CHANGE](#) command. In this mode, when the length of the from/to strings in a [CHANGE](#) command are different, the presence or absence of columns in the data has no effect; the strings are changed and the data to the right of the change shifts left or right accordingly.

The current mode in which SPFLite is operating is associated and retained as another file Profile property, and will be displayed in the [Status Bar](#) at the bottom of the screen. The default shift mode for a Profile can be changed at any time with a **CHANGE DS** or **CHANGE CS** command.

As well, the shift mode for an individual [CHANGE](#) command can be explicitly specified by adding a [CS](#) or [DS](#) operand to the [CHANGE](#) command.

Starting Point and Direction of the Search

To control the starting point and direction of the search, use one of the following operands. SPFLite sometimes describes these keywords as placement operands, since they describe the place where the search starts and the place toward which it is going.

NEXT	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of string-1. NEXT is the default and is generally not specified.
ALL	Starts at the top of the data and searches ahead to find all occurrences of string-1. When complete, a message is issued stating the number of occurrences found. If you use this operand with CHANGE , the lines changed are marked with ==CHG> flag. The status of these lines can be changed to normal by RESET . When used with FIND then <u>all</u> occurrences of the string will be hi-lighted in the text.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of string-1.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of string-1.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of string-1.
LEFT	A LEFT operand causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the left-most occurrence of search-string is found/changed, and any other instances on that same line are ignored.
RIGHT	A RIGHT operand causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the right-most occurrence of search-string is found/changed, and any other instances on that same line are ignored.

If you specify **ALL** or **FIRST**, the direction of the search is forward. When you press the assigned function keys, the [RFIND](#) or [RCHANGE](#) commands find or change the next occurrence of the designated string. If you specify **LAST** or **PREV**, the direction of the search is backward. When you specify those operands, the editor finds or changes the previous occurrence of the string. The search proceeds until the editor finds one or all occurrences of string-1, or the end of data.

When **ALL** is specified, the **FIND** and **CHANGE** commands will report the **number of lines** in which a string is found or changed, in addition to the **number of times** the string itself is found.

If you omit the **ALL** operand on the **CHANGE** command, the editor searches only for the first occurrence of string-1 after the current cursor location. If the cursor is not in the data area of the panel, the search starts at the beginning of the first line currently displayed. Scrolling is performed, if necessary, to bring the string into view.

Note: The **SPLIT** command allows an extended syntax of **ALL FIRST** and **ALL LAST**. See [SPLIT - Split Lines Using Find/Change Strings](#) for more information.

Scroll Alignment using TOP

When a command such as **FIND** or **CHANGE** finds successive lines on the same screen, SPFLite will "walk" down the screen by just repositioning the cursor, and will only scroll the screen when there are no more lines to be found on that same screen. If you include the word **TOP** in your command, each time a successive line is found, the screen will be repositioned so that the found line appears on the top of the screen. This can be useful when scrolling through large files of repetitive fields, so that scrolling doesn't make the screen "jump around".

You can use the **TOP** keyword on the commands **APPEND**, **CHANGE**, **COMPRESS**, **DELETE**, **EXCLUDE**, **FIND**, **FLIP**, **JOIN**, **LINE**, **LOCATE**, **NDELETE**, **NEXCLUDE**, **NFIND**, **NFLIP**, **NREVERT**, **NSHOW**, **NULINE**, **PREPEND**, **REVERT**, **SHOW**, **SPLIT**, **TAG** and **ULINE**.

Qualifying the Search String Context

You can specify the "search context" of string-1 by using the operands **PREFIX**, **SUFFIX**, **WORD** or **CHARS**. The search context defines "where" or "under what circumstances" a given search string is considered to be "found".

PREFIX

Locates string-1 at the beginning of a word. String-1 must be at the beginning of the line, or must be preceded by a non-WORD character. **PREFIX** may be abbreviated as **PRE** or **PFX**.

SUFFIX

Locates string-1 at the end of a word. String-1 must be at the end of the line, or must be followed by a non-WORD character. **SUFFIX** may be abbreviated as **SUF** or **SFX**.

WORD

A **WORD** string must follow the rules for **PREFIX** and **SUFFIX** at the same time: It must be at the beginning of the line, or must be preceded by a non-WORD character, **and** it must be at the end of the line, or must be followed by a non-WORD character.

CHARS

String-1 is searched for as-is without regard to what precedes or follows it.

CHARS is the default, and so the keyword **CHARS** is not normally used. **CHARS** is allowed primarily for ISPF compatibility purposes, and when the default has been changed. This default can be configured to either **CHARS** or **WORD** in the Global Options dialog. See [Options - General](#) for more information.

In the following examples, the editor would find the noted strings only:

FIND 'DO'	- <u>DO</u> <u>DONT</u> <u>ADO</u> <u>ADOPT</u> 'DO' (DONT)	Finds all of them
FIND 'DO' CHARS	- <u>DO</u> <u>DONT</u> <u>ADO</u> <u>ADOPT</u> 'DO' (DONT)	SAME THING -
Finds all of them		
FIND PREFIX 'DO'	- DO <u>DONT</u> <u>ADO</u> <u>ADOPT</u> 'DO' (DONT)	Finds DONT (DONT)
FIND SUFFIX 'DO'	- DO <u>DONT</u> <u>ADO</u> <u>ADOPT</u> 'DO' (DONT)	Finds ADO
FIND WORD 'DO'	- <u>DO</u> <u>DONT</u> <u>ADO</u> <u>ADOPT</u> 'DO' (DONT)	Finds DO and 'DO'

Truncation following the CHANGE

By specifying the keyword **TRUNC** in the **CHANGE** command, you can request that all line data **following** the new change string be deleted from the line. Because **TRUNC** is a new reserved word, you would have to quote this word if you ever wanted to use it as a normal string value in **CHANGE** command.

The following example changes the characters "END) " to "END ." and deletes all remaining characters on the line.

```
CHANGE "END) " "END ." TRUNC
```

which would alter the line

```
COMING TO AN END) BUT NOT BEFORE
```

into

COMING TO AN END .

You can use truncation to truncate all characters *including* the search string by making the change string of length zero:

CHANGE ") " " TRUNC

which would alter the line

COMING TO AN END) BUT NOT BEFORE

into

COMING TO AN END

Delimiters used to determine Word, Prefix and Suffix boundaries

In order to determine what a 'word', 'prefix' or 'suffix' actually is, SPFLite uses a set of characters to determine what a valid WORD is. The characters that make up a WORD are specified in the Profile's [WORD](#) control string. These default characters are:

A-Z a-z 0-9

However, some computer languages allow other characters to be used in variable names (such as the _ underscore character in many languages, the - dash character in COBOL programs, or the \$, # and @ in PL/1). This can cause **word** searches to find strings which *you* don't consider to be a "word", or it might *fail* to find words you *do* want to be words. SPFLite allows you to modify the list of valid WORD characters. The modified list you create will be associated with the file type being edited, and will be saved and used in future edit sessions of this file type automatically. See the WORD line command, and [Working with Word and Delimiter Characters](#) for details on how to change these characters.

Note that the space character is always assumed to be a delimiter. This assumption cannot be changed.

If **you** are the one looking for a string, and you consider it to be a "word", how could SPFLite not treat it as a word and find it? For example, if you were looking for a word ABCD, and you said **FIND ABCD WORD**, it seems pretty straight-forward that if ABCD exists as a word, it would get found. However, what if you were looking for **any** four-character word? If the word only had English letters, you could be pretty certain you'd find it with **FIND P'@@@@' WORD**.

But suppose it were a four-character string that might have an underscore or dash in it; what then? Just saying **FIND P'====' WORD** won't work, because you would find **any** four characters, as long as there were delimiters next to it, and that's not what you wanted. The data you found could be well-delimited **junk**. How do you solve this problem, and only find what you really want - and nothing else?

First, you modify the WORD lines so that your set of valid WORD characters is correct. That "expands" the definition of a "word character" and removes those characters as delimiters. For example, if you add the underscore to the WORD characters, then it will no longer be considered as a character that delimits a word.

Then, how can you tell SPFLite to only look for the characters that **you** say are part of a special kind of "word" that you want? We could have changed how the @ picture works, but it's best not to tamper with that, so @ stays as-is, and only matches letters. Instead, SPFLite defines two extended picture code types:

& defines any character in the set of WORD characters
% defines any character **not** in the set of WORD characters

So, if you want a four-character string of **your kind of words**, as defined by your settings of the WORD string, you would do this:

FIND P'&&&&' WORD

This works because WORD means a string delimited by **non-WORD** characters or the edges of a line, and the **&** picture means all characters which are **in** in the WORD character list.

What is nice about this is that the WORD characters are in the PROFILE, so whether you have ordinary text, C programs, COBOL programs, or some special-purpose data, you can customize each file type to exactly the definition of what a "word" means to best suits **your** needs.

Note: While **p** ' & ' can be used to find "your kind of words", SPFLite does not look more closely than that. So, there is no provision to search for things like "your kind of words in lower case only". That restriction only makes sense, because if you were to add special characters like \$, # or @ to the list of "word" characters, are they upper case or lower case? Ordinarily, the answer would be "neither", and since SPFLite doesn't know what your intentions are if it's other than that, it can't guess. If you really apply such fine distinctions to your data, you may need to write a programmable macro to inspect your data as needed.

Limiting the search to specific columns

The col-1 and col-2 operands allow you to search only a portion of each line, rather than the entire line. These operands, which are numbers separated by at least one blank, show the starting and ending columns for the search. The following rules apply:

- If you specify neither col-1 nor col-2, the search continues across all columns within the current boundary columns.
- If you specify col-1, the editor finds the string only if the string starts in the specified column.
- If you specify both col-1 and col-2, the editor finds the string only if it is entirely within the specified columns.

Limiting the search to Excluded or Non-Excluded lines

You can limit the lines to be searched by using the **X** or **NX** operands:

X means search only excluded lines
NX means search only unexcluded lines

A number of commands also allow you to control the exclusion status of a line after it is found or changed, using the **MX** (make excluded) or **DX** (do not change exclusion status) keywords.

Limiting the search to User or non-User lines

You can limit the lines to be searched by using the **U** or **NU** operands:

U means search only user lines
NU means search only non-user lines

All data lines are either User lines (U lines) or non-User lines (also called V lines). A user line is marked by a vertical bar in the "gap column" to the right of the sequence number field.

A line can be made a User line by the primary commands **ULINE** and **NULINE** and the line command **U/UU**. A line can be made a non-User line by the primary commands **REVERT** and **NREVERT** and the line command **V/VV**.

Limiting the search to specific highlighted strings

You can request searches only locate strings that have previously been highlighted in specific colors. This is done

via use of the color-selection-criteria keywords. These can be any of the names shown in ["Options - Hi-Lites"](#) like **BLUE, GREEN, BLACK etc...** More details can be found in ["Color-Selection-Criteria-Specification"](#).

Repeating the **FIND** and **CHANGE** commands

The easiest way to repeat **FIND**, and **CHANGE** commands, without retyping them, is to assign those commands to function keys. There are already ISPF-compatible key-mapping defaults made at installation time to do this, using the **RFIND** and **RCHANGE** commands:

F5 RFIND
F6 RCHANGE

The search begins at the cursor. If the cursor has not moved since the last **FIND**, or **CHANGE** command, the search continues from the string that was just found. Instead of retyping string-1, you can type an ***** **asterisk** to specify that you want to use **the last search string**.

If you decide to type **RCHANGE** or **RFIND** on the Command line instead of using a function key, position the cursor at the desired starting location before pressing Enter. If you are searching for every string in a file, one at a time, from beginning to end, and you are using the mapped functions F5 and F6, the cursor will already be in the location you need. You would only have to reposition it if you have moved it through some type of editing action. (That's why most people use F5 and F6, for that very reason - it's easy and convenient.)

All these commands share the same string-1. Therefore:

FIND ABC

followed by:

CHANGE * XYZ

first shows you where ABC is, and then replaces it with XYZ. However, you can do this more easily by typing:

CHANGE ABC XYZ

Then press F5 to repeat **FIND**. The editor finds the next occurrence of ABC. You can either press F5 to find the next ABC, or F6 to change it. Continue to press F5 to find remaining occurrences of the string.

The previous value of a search string, specified by an asterisk or by use of **RFIND** or **RCHANGE**, is retained until you end your editing session.

SPFLite adds a new command **RLOCFIND**, which repeats the last **LOCATE** or **FIND** command, whichever has been done most recently. For many users, it will be more productive to assign **RLOCFIND** to F5, which can now serve both as a repeat-find (**RFIND**) and as a repeat-locate (**RLOC**) command.

RFIND, RLOCFIND and **RCHANGE** also apply to the **SPLIT** and **JOIN** primary commands, and as well applies to the **DELETE** primary command, when **PREV** or **NEXT** is specified, or when **NEXT** is implied by the absence of other keywords.

Case-Conformant Change Strings

When you do a change with search string having a type code of T, it matches letters in a case-insensitive way. So, if you say,

CHANGE WORD T'four' 'nine'

it will match on the word "four" no matter how it is capitalized. However, regardless of the original string, the result of this **CHANGE** will always be capitalized as "nine":

four becomes **nine**
Four becomes **nine**

FOUR becomes **nine**

and so, the result fails to *conform* to the character-casing of the original string.

With *case-conformant changes*, you can make the result string match the pattern of upper and lower casing that existed in the original string. That is, **now** you can make *this* happen:

four becomes **nine**
Four becomes **Nine**
FOUR becomes **NINE**

How? Just put a type code of T on the change string, like this:

CHANGE WORD T'four' T'nine'

That's great if the two strings are the same size, but what if they're not?

In the easy case, when the change string is *shorter*, the pattern of upper and lower casing applies for as long as the change string is. If you change a 4-letter word into a 3-letter word, the first 3 positions of the search string are used as the "capitalization pattern", like this:

CHANGE WORD T'four' T'two'

four becomes **two**
Four becomes **Two**
FOUR becomes **TWO**

When the change string is *longer*, it's a little more complicated. The pattern of upper and lower casing applies for as long as the search string is. Beyond that point, the case of the last character in the search string is used as a guide to *propagate* the casing of the result string from that point forward. (This "remaining result string" could be called the 'tail' of the result string.)

What happens if the last character of the search string isn't a letter? SPFLite uses the following policy to handle this:

- Characters of the search string are scanned from right to left, starting with the last character, until a letter is found or the beginning of the string is reached.
- If a letter is found by this scan, the case of that letter is used as a guide to *propagate* the casing of the tail of the result string.
- If a letter is not found by this scan, the casing of the tail of the result string is copied in lower case.

Is that a good choice? It's a toss-up. In designing this, five or six *different* possible approaches were considered, and many were hard to explain and harder to implement. For most users, the propagation rules are about as good as any. For changing one English word to another, it works well. For changing strings like alphanumeric-coded values (such as part numbers) it may or may not be the answer for everyone. See the discussion below in case these rules are not what you need.

If you change a 4-letter word into a 5-letter word, the first 3 positions of the search string are used as the capitalization pattern of the first 3 positions of the change string, and position 4 of the search string is used as the pattern for everything else:

CHANGE WORD T'four' T'seven'

four becomes **seven**

Four becomes **Seven**
FOUR becomes **SEVEN**

Here, the R of **FOUR** is used as the pattern for positions 4 and 5 of the string **SEVEN**. For the first two, the R is lower case, so E and N become lower case. In the last one, the R is upper case, so the E and N become upper case.

A few final notes:

- It is not a requirement that the search string have a type code of T, if **CASE T** is in effect. But, if you have a type code of C or **CASE C** is in effect, you will always find the same string, cased in the same way. You *could* do that, but there wouldn't be much point to it.
- As you might expect, the case of the actual **CHANGE** command operands for case-conformant changes is not significant. That means,

CHANGE WORD T'four' T'seven'
and
CHANGE WORD T'FOUR' T'SEVEN'

will work exactly the same way. And, of course, the T itself is case-insensitive.

- When **CASE T** is in effect, it implies type code T on the *search* string, unless you explicitly say otherwise. For the *change* string, type code T (and thus, a case-conformant change) is *never* assumed, even when **CASE T** is in effect. You have to put the T code on the change string yourself to get a case-conformant change.

You might ask, what if SPFLite's rules for Case-Conformant strings aren't good enough for my needs? You basically have these choices:

- Use SPFLite's Case Conformant strings for what they do, and if you need to, 'correct' any improper casing after the fact. That might be a good choice if the **CHANGE** did what you wanted most of the time, and you just had to "touch up" a few exceptions.
- If Case Conformant strings do something you really dislike, you can **UNDO** the change or **CANCEL** the edit session and try something else

Changing the Default Search Context

When a **FIND**, **CHANGE** or similar command is used, and none of the operands **CHARS**, **WORD**, **PREFIX** or **SUFFIX** are specified, the **FIND** or **CHANGE** command will normally assume that the string being searched for is a CHAR string; that is, any delimiters next to the search string are ignored. This is the standard way SPFLite and ISPF look for strings.

If you wish to look for **WORD** strings, that is, strings with a delimiter on each side, you normally would specify the **WORD** operand, as in **FIND ABC WORD**.

You are now able to set the default search context to either **WORDS** or **CHARS**. When set to **WORDS** mode, every **FIND**, **CHANGE** or similar command that can take the **WORD** operand will assume it has already been set.

When the default search context is set to **WORDS**, a **W** will appear the C/T indicator on the status line that shows the current CASE C/T mode. Thus, depending on the CASE mode, the indicator will show either **C W** or **T W** on the status line when you are in WORD mode. When it is set back to **CHARS** mode, the indicator will show either **C** or **T** on the status line, without the **W**.

If you find you frequently need to search for string in WORD mode, you can configure SPFLite to always use WORD mode as a default. This is done by checking the entry [Use WORD as the default for FIND/CHANGE commands](#). When this checkbox is unchecked, SPFLite will look for strings as CHAR values as usual. See "Options - General".

SPFLite Global Options

<input type="button" value="General"/> <input type="button" value="FM"/> <input type="button" value="Submit"/> <input type="button" value="Screen"/> <input type="button" value="KBD"/> <input type="button" value="Status"/> <input type="button" value="Schemes"/> <input type="button" value="HiLites"/>	
<input checked="" type="checkbox"/> Audible BEEP on Errors? <input checked="" type="checkbox"/> Visual BEEP on Errors? <input checked="" type="checkbox"/> Delete to Recycle Bin? <input type="checkbox"/> Use DIR name AS PROFILE when no File extension? <input type="checkbox"/> Use WORD as the default for FIND/CHANGE commands? <input checked="" type="checkbox"/> Allow 2-D mouse selection without Shift/Ctrl/Alt? <input type="checkbox"/> Only English letters A-Z and a-z are considered alphabetic <input type="checkbox"/> Default RESET will Revert User line status.	
Minimum command length for RETRIEVE?	<input type="text" value="3"/> <input type="button" value="▼"/>
Command separator char.	<input type="text" value=";"/>
Use fast renum if # lines > this value	<input type="text" value="999999"/>
Default # columns for data shifts (Min 1)	<input type="text" value="4"/>
Numer of columns/lines per Auto-Scroll	<input type="text" value="2"/>
Notify tabs on external file change	<input type="text" value="All"/> <input type="button" value="▼"/>
Line Commands repeat limit (0=no limit)	<input type="text" value="0"/>
Normal characters for screen display and for P.' picture literals <code>!"#\$%&'()*+,./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz</code>	
Display Char. for Invalid characters.	<input type="text" value="~"/>

 INI File is:

If you want to quickly change the search context from the edit command line without using the Global Options window, you can issue a **FIND WORDS** or **FIND CHARS** command. See [FIND - Find a Character String](#) for more information.

If you issue a **RESET** command with no operands, the default search context will revert to the setting you have for this checkbox, either **WORD** mode (checked) or **CHAR** mode (unchecked).

Restrictions on the use of LAST and PREV with Regular Expressions

When a **FIND** or **CHANGE** search operand is a Regular Expression (a string with an **R** type code) and reverse-order searching is done with **PREV** or **LAST**, only the left-most occurrence on any given line is found, as if the **LEFT** operand had been used. That is, the command

```
FIND R'ABC' PREV
```

is treated as if it were specified as

```
FIND LEFT R'ABC' PREV
```

and

```
FIND R'ABC' LAST
```

is treated as if it were specified as

```
FIND LEFT R'ABC' LAST
```

This limitation stems from the regular-expression engine used by SPFLite.

Specifying a Picture or Format String

Contents of Article

[Introduction](#)

[Picture Strings](#)

[Special note for P'\[" and P'" or P'{ and P'}](#)

[Picture String Examples](#)

[Using Picture strings as the TO operand of a CHANGE command](#)

[SPFLite Format Change Strings](#)

[Displayable and non-displayable characters](#)

Introduction

IBM ISPF users are familiar with Picture strings. These are used to perform pattern-matching for the search-string part of **FIND** and **CHANGE** commands, and for selectively changing text to upper-case or lower-case in the change-string part of **CHANGE**. Pictures are like a simple form of regular expression. When IBM first introduced them in ISPF, they were a novel and innovative feature for their time, but by today's standards they may seem quite rigid and limited. Still, because Pictures are easier to use than regular expressions, they remain useful. And, as implemented in SPFLite, they are far more useful than IBM's. SPFLite's implementation of Picture strings is more extensive than in ISPF, and includes support for the Picture-like **Format** string.

A picture or format string in a **FIND**, **CHANGE**, **EXCLUDE**, **SPLIT** or **JOIN** command allows you to search for a particular category of character, rather than looking for a specific literal character value.

Picture and Format strings are very similar. However, **Format** strings may only be used as the string-2 change-string operand in **CHANGE**, **SPLIT** and **JOIN** commands.

The **SPLIT** and **JOIN** primary commands use some additional special characters.

New variations on **P'!'** now also allow you to copy arbitrary text and perform case conversion at the same time.

When a **FIND** or **CHANGE** search picture uses the special characters **=** or **.** and a character that cannot be displayed is found, that character's hexadecimal representation is used in the confirmation message that appears in the upper-right corner of the Edit or Browse screen. For example, the command **FIND P'..'** could result in the message **CHARS X'0415' found**. This change was made to be compliant with how ISPF handles this situation.

Picture Strings

You can use special characters within the picture string to represent the kind of character to be found, as follows. Note that SPFLite does not support the ISPF APL pictures **P'_'** and **P'÷'**, and will treat these as ordinary data characters.

P'=' Any character

P'^' or **P'¬'** Any character that is not a blank. Both **P'^'** and **P'¬'** work identically.

Note that **¬** can be either the ANSI **¬** character (X'AC') or the OEM **¬** character (X'AA'). You should select the character that displays properly with your chosen editor font, but either one will work.

<code>P' #'</code>	Any numeric character, 0-9 Note that the three special superscript characters in ANSI for ¹ , ² and ³ are not considered numeric.
<code>P' -'</code>	Any non-numeric character, which is any character not matching a Picture of <code>P' #'</code>
<code>P' .'</code>	Any non-displayable character; that is, any character not in the Normal list in Global Options
<code>P' @'</code>	Any alphabetic character, uppercase or lowercase. This includes the non-English letters in the ANSI character set.
<code>P' <'</code>	Any lowercase alphabetic character. This includes the non-English letters in the ANSI character set.
<code>P' >'</code>	Any uppercase alphabetic character. This includes the non-English letters in the ANSI character set.
<code>P' \$'</code>	Any special character, neither alphabetic nor numeric.
<code>P' %'</code>	Any character not defined in the WORD setting, i.e. a delimiter.
<code>P' &'</code>	Any character defined in the WORD setting. A span of <code>P' &'</code> characters would be a "user-defined word".
<code>P' !'</code>	Allowed only in the <i>change-string</i> . It represents the entire string value actually found (not the string-1 <i>operand</i>), regardless of its length. It is allowable to use the <code>!</code> code multiple times in the change-string; if done, the found string is repeated in the result as many times as the <code>!</code> code is used. The found string is copied as-is without case conversion.
<code>P' !='</code>	A Picture of <code>P' !'</code> may be specified as <code>P' !='</code> with essentially identical meaning. <code>P' !='</code> would only be used in cases where a Picture of <code>P' !'</code> needed to be followed by a Picture of <code>P' >'</code> or <code>P' <'</code> to mean that (a) a found string is copied as is, and then (b) an individual copied character is converted to upper or lower case. The <code>P' !='</code> notation is like an "escape" that prevents the <code>!</code> from being confused with <code>P' >'</code> or <code>P' <'</code> .
<code>P' !>'</code>	Allowed only in the <i>change-string</i> . It represents the entire string value actually found (not the string-1 <i>operand</i>), regardless of its length. It is allowable to use the <code>!></code> code multiple times in the change-string; if done, the found string is repeated in the result as many times as the <code>!</code> code is used. The found string is copied and converted to upper case.
<code>P' >'</code>	If desired, <code>P' !>'</code> can be shortened to <code>P' >'</code> if the <code>></code> character is mapped to some key, so that it can be typed on the command line. Note that <code>></code> can be either the ANSI <code>></code> character (X'BB') or the OEM <code>></code> character (X'AF'). You should select the character that displays properly with your chosen editor font, but either one will work.
<code>P' !<'</code>	Allowed only in the <i>change-string</i> . It represents the entire string value actually found (not the string-1 <i>operand</i>), regardless of its length. It is allowable to use the <code>!<</code> code multiple

times in the change-string; if done, the found string is repeated in the result as many times as the ! code is used.

The found string is copied and converted to lower case.

P' <<'

If desired, **P' !<** can be shortened to **P' <<** if the << character is mapped to some key, so that it can be typed on the command line.

Note that << can be either the ANSI << character (X'AB') or the OEM << character (X'AE'). You should select the character that displays properly with your chosen editor font, but either one will work.

P' ['

In a search string, **P' ['** is used to represent the left-hand edge of a line. When used this way, the [code must appear left-most in the Picture string, and there must not be any other [codes present in the Picture, unless escaped as \[. Because the [code represents the edge of a line, and not an actual data character value, there must be at least one other character in the string, such as **P' [ABC'**. A find-string written literally as **P' ['** is illegal.

In a *change-string*, a Picture or Format of **P' ['** is treated as ordinary data.

P'] '

In a search string, **P'] '** is used to represent the right-hand edge of a line. When used this way, the] code must appear right-most in the Picture string, and there must not be any other] codes present in the Picture, unless escaped as \]. Because the] code represents the edge of a line, and not an actual data character value, there must be at least one other character in the string, such as **P' ABC] '**. A find-string written literally as **P'] '** is illegal.

In a *change-string*, a Picture or Format of **P'] '** is treated as ordinary data.

P' { '

In a search string **P' { '** is used to represent the left-hand edge of a line **including any optional leading blanks**. When used this way, the { code must appear left-most in the Picture string, and there must not be any other { codes present in the Picture, unless escaped as \{. Because the { code represents the logical left edge of a line, and possibly not an actual data string (since the optional blanks may not be present), there must be at least one other character in the string, such as **P' {ABC'**. A **find** string written literally as **P' { '** is illegal. When a successful find occurs, the effective length of the 'found' string includes the leading spaces, if present.

In a **change** string, a Picture or Format of **P' { '** is legal, and is treated as a variable number of spaces equal to the leading spaces found by the accompanying *search-string*, and may be of zero length. The change Picture or Format of **P' { '** is somewhat similar to the change Picture or Format of **P' ! '** in that it represents a variable number of characters, and can appear anywhere in the change string, even multiple times. When the { code is used in a change string Picture or Format, and there is no associated { code in the find picture, the { code on the change side represents a zero-length string.

P' } '

In a search string **P' } '** is used to represent the right-hand edge of a line **including any optional trailing blanks**. When used this way, the } code must appear right-most in the Picture string, and there must not be any other } codes present in the Picture, unless escaped as \}. Because the } code represents the logical right edge of a line, and possibly not an actual data string (since the optional blanks may not be present), there must be at least one other character in the string, such as **P' ABC}**. A **find** string written literally as **P' } '** is illegal. When a successful find occurs, the effective length of the

'found' string includes the trailing spaces, if present.

In a **change** string, a Picture or Format of `P' } '` is legal, and is treated as a variable number of spaces equal to the trailing spaces found by the accompanying *search-string*, and may be of zero length. The change Picture or Format of `P' } '` is somewhat similar to the change Picture or Format of `P' ! '` in that it represents a variable number of characters, and can appear anywhere in the change string, even multiple times. When the `}` code is used in a change string Picture or Format, and there is no associated `}` code in the find picture, the `}` code on the change side represents a zero-length string.

`P' | '`

In the *change-string* Picture of a **SPLIT** primary command, `P' | '` is used to represent a line-break, where the line is to be split in two. For **SPLIT**, the contents of the change-string completely replace the find-string. It is possible to delete the entire find-string and replace it by a line-break by having a change-string written literally as `P' | '` as the only Picture code present.

For commands other than **SPLIT**, a *change-string* Picture containing `P' | '` is ignored, and will neither be used for line-splitting purposes nor as a literal `| '` character. If you wish to have a change-string Picture contain a literal `| '` character, it must be escaped as

`P' \\ | '`

`P' \x'`

The \ backslash is used to escape a character that might otherwise be used as a special Picture code, like `$`. For example, `P'@#'` is used to find to a letter followed by a digit, whereas `P'\\@#'` is used to find to an `@` sign followed by a digit. When \ backslash is doubled as `\\` or is used as the last character of a Picture string, it is taken literally as a single \ backslash data character. When \ backslash precedes a letter, the letter is taken literally; that is, it matches exactly in case, even when **CASE T** is in effect.

Special note for `P'["` and `P'"]` or `P'{"` and `P'}"`

It is possible to specify both `[` and `]` (or `{` and `}`) in the same find-string Picture, when `[` begins the Picture, and `]` ends the Picture; that is, if you specify a Picture in the form `P' [string] '`. When this is done, it requests the command to search for a string which is the only data present on a line. For example, the find-string Picture `P' [ABC] '` will find the string ABC when it is the only data present on a line, with neither leading nor trailing spaces present.

The `P' {ABC} '` version would find ABC when it is the only data on a line, regardless of any leading or trailing spaces. **Note:** When **HILITE FIND** is effect, a find picture like `P' {ABC} '` will cause any extra spaces on the edge of the line that correspond to a `{` or `}` Picture to be included in the string that is highlighted.

For the **JOIN** command, the first string operand must be a Picture string of the form `P' [string] '` or `P' string] '`.

The first string operand of **JOIN** can also be a Picture string of the form `P' {string} '` or `P' string} '`.

It is not possible to directly join one line to **both** the line that precedes it **and** the line that follows it, at the same time. That is to say, a **from-string** of a **JOIN** command cannot be specified in the form of `P' [string] '` or as `P' {string} '`. For any given line, only one **JOIN**, on one side of a line, is allowed. **JOIN** cannot perform a left-join and a right-join at the same time, since this would imply converting three lines of data into one line in a single join operation, an action that SPFLite does not support.

A find-string written literally as `P' [] '` is illegal. This cannot be used to find zero-length lines, because there is literally no data to find.

Even though the new Picture codes of { and } represent optional spaces, a find-string written literally as `P'{}'` is also illegal because, based on the definition of these codes, one possible meaning of such a Picture is a string of zero length, which cannot be found.

To find a zero-length line, the command `NFIND P'='` or `LOCATE SIZE 0` may be used.

Picture String Examples

- To find a string of 3 numeric characters:

`FIND P'###'`

- To find any 2 characters that are not blanks but are separated by a blank:

`FIND P' ^ ^'`

- To find a blank followed by a numeric character:

`FIND P' #'`

- To find a numeric character followed by AB:

`FIND P' #AB'`

- To find the next character in column 72 that is not a blank:

`FIND P' ^ 72`

- To change any characters in columns 73 through 80 to blanks:

`CHANGE ALL P'=' ' ' 73 80`

- To find the next line with a blank in column 1 and a character in column 2 that is not a blank:

`FIND P' ~' 1`

Using Picture strings as the TO operand of a CHANGE command

SPFLite expands on the ISPF **CHANGE** Picture standard in a number of ways that make them even more useful:

- ISPF allows three types of “copy codes” in a change Picture. The = sign is used to copy a character as-is from the search string to the result string; a < sign will copy data and convert it to lower case, and a > sign will copy data and convert it to upper case. SPFLite adds a **fourth** “copy code” of ~ **tilde**. A tilde in a change Picture is used to “match against” a character in the search string, but then it will “ignore” or “discard” that character in the result string. So, a change Picture of `P'=~='` will copy the first and third characters of the search string, but will skip over the second character of the search picture. The resulting string value will be 2 characters long.
- A second extension is the picture character ! which is taken to mean the **entire string** located by the **CHANGE** function. It is *not* the string-1 operand of CHANGE, but is the actual value found by the command. (So, if the search operand were `P'##'` then the actual value found is not '#' but *might* be a '5'. It can be used multiple times in the change string if desired, to replicate the found string. If a command `CHANGE P'@@@' P'!!!'` were issued, and it located the data string ABC, the resulting string would be ABCABCABC.
- You can use change Pictures and Formats of the form `P'!>'`, `F'!>'`, `P'!<'` and `F'!<'`. These work like `P'!!'` and `F'!!'` do, with the added effect of converting the result to upper or lower case afterwards. See the discussion above for details.
- The ISPF standard required that a change Picture must have exactly the same length as the search string

. That prevents Pictures from being used to shorten or lengthen the value found by the search string. SPFLite relaxes this requirement. Now, a change Picture can be shorter or longer than the search string. When this is done, SPFLite does impose one requirement: If the change Picture is longer, the positions of the change Picture that extend beyond the length of the search string cannot contain “copy codes” of = < > or ~ because there is nothing to match against. For example, you can't say **CHANGE P'@@@' P'@@@= WORD ALL**, because the fourth position of the change string has an = sign, asking that the fourth character of the find string be copied. But, there **is** no fourth character, since the command only asked to look for strings of length 3.

- In ISPF, pattern codes like \$, # and @ are allowed in search Pictures, but not in change Pictures, even though they don't really mean anything special in a change Picture. IBM just doesn't allow it. SPFLite does. Anything in a change Picture which is not one of the “copy codes” is treated as ordinary data. As noted, SPFLite allows an extended copy code of !, !> and !< to represent the entire found string.
- SPFLite introduces the [CASE](#) primary command, which may be issued as **CASE T** or **CASE C**. When you issue this command, you will see a **T** or **C** on the status line in a little box by itself. This defines what the default case sensitivity mode is for quoted strings that don't have a T or C type code on them. In ISPF, IBM chooses to treat alphabetic characters in search Pictures as case-insensitive (like T strings), and they offer no way to change that assumption. In SPFLite, the current state of **CASE** defines how search Pictures are treated. So, a *search* Picture of P'ABC' is case-*insensitive* if there is a **T** in the status line, and is case-*sensitive* if there is a **C** there.

Note that **CASE T** or **CASE C** affects how a *search* string is handled. The *change* string in a **CHANGE** command, no matter what string type, is treated as case-sensitive, in the sense that any alphabetic data you enter is used exactly as you typed it, and **CASE T/C** is ignored there.

- In ISPF, any character defined as a Picture code cannot be used as ordinary data. In SPFLite, you can escape any special character with a \ backslash, including the backslash itself.

SPFLite Format Change Strings

With SPFLite, a new Picture-like change string is now available: the *Format* string. A Format can only be used on the *change*-side of a **CHANGE** command; it cannot be used as a *search* string on **FIND**, **CHANGE** or any similar command. Format change strings (we'll just call them *Format* strings from now on) have the following properties:

- Format strings have a type code of **F** rather than the type code **P** for Pictures.
- Like Pictures, a Format string can be longer or shorter than the search string.
- The same special codes of = < > and ~ allowed in a change Picture are allowed in a change Format, and they have the same basic meaning. Also, the same characters considered as ordinary data in a change Picture are ordinary in a Format.
- Since a Format can only be used for changes, it is not affected by the state of **CASE T** or **CASE C**.

The thing that makes change Formats different than change Pictures is how the “copy codes” of = < > and ~ are “lined up” or “associated” with the search string. Here is the distinction:

- In a change **Picture**, a copy code of = < > or ~ in position **n** is used to match up to a character from the search string *in that same position n*. So if = is in position 3, it copies position 3 of the search string into position 3 of the result string.
- In a change **Format**, each copy code of = < > or ~ is effectively “enumerated” from left to right. That is, if you look at the Format string, and *ignore* any characters which are *not* a copy code of = < > or ~, then the *first* copy code is “code 1”, the *second* code is “code 2”, etc. *regardless of what relative position they are in*. Then, the *first* copy code in the Format is matched up to *position 1* of the search string; the *second* copy code in the Format is matched up to *position 2* of the search string; etc.

Why use Format strings? With all the features of Picture strings, you might think that was enough. You can even insert data to the right of a found-string using a change Picture longer than the search string.

But, what if you wanted to insert something *before* the search string? With Pictures, you can't do it.

Example: I want to find all 3-letter words and put parentheses around them. Let's try it with a Picture:

```
CHANGE P'@@@' P'(==)' WORD ALL
```

However, this won't work. Why not? Let's match up the two strings, and see what happens. The arrows represent character matching, and the dashes represent inserted data:

```
P'@@@'  
-^A^-  
P'(==)'
```

Here, you can see where the **A** red arrow is that the third = is being matched up against a non-existent search string position, and that's illegal, even in SPFLite.

Let's try this again, using a change Format:

```
CHANGE P'@@@' F'(==)' WORD ALL
```

This *does* work. Why? Again, let's match up the two strings, and see what happens:

```
P'@@@'  
-^A^-  
P'(==)'
```

Here, the *second character* in the Format is the *first copy code*. Enumerating the copy codes, that corresponds to *position 1* of the search string. The same holds for the other = signs. So, if CHANGE finds a word like **ABC**, it will change it to **(ABC)** and **XYZ** will be changed to **(XYZ)**.

Suppose you wanted to change a word like **ABC** into **(A/B/C)**? Easy. Just put the slashes in the format to insert them:

```
CHANGE P'@@@' F'(=/=/)' WORD ALL
```

Note that in the Format, the = signs are in relative positions 2, 4 and 6, but since we *enumerate* them, and we aren't looking at their *positions* in the Format string, the = sign in position 2 is the *first = sign*, so it copies position 1; the *second =* in position 4 copies position 2, and the *third =* in position 6 copies position 3.

Any time you need to insert data *before* or in the *middle* of the search string, Format is what you want. If you want to insert *after*, either Format or Picture should work fine.

Note: When a Format string contains the "copy entire string" codes of **F'!!'**, **F'!>'** or **F'!<'**, the copying operation for these codes is done **independently** of any copy codes of = < > or ~ that may be present in the Format string. The **F'!!'**, **F'!>'** or **F'!<'** codes do not count when considering the enumeration of characters in the found-string.

Why didn't we just make Pictures work the same as Formats, and be done with it? Because, for people who make extensive use of Pictures, Formats don't really work exactly the same way as Pictures, and we didn't want to cause confusion or break old editing habits if it wasn't necessary to do so.

Disposable and non-displayable characters

The picture character `P' . '` (period) requests a search for non-displayable characters. Depending on the font you are using, particularly when national characters are involved, deciding what is non-displayable can be confusing. SPFLite provides you with the ability to specifically indicate what characters you wish to consider displayable.

The Options - General settings page allows you to alter what these characters are. See [Options - General](#) for how this is changed.

Note that the Normal Characters on that screen define what the `P' . '` picture does **not** match to. Be careful not to get this point confused.

The non-displayable picture concept comes from IBM's ISPF that originally ran on 3270 terminals, which have far fewer displayable characters in EBCDIC than PCs do using the ANSI character set. A 3270 terminal would "lock up" if non-displayable characters were written to the screen, but this will not happen in SPFLite. Because of this, the issue of non-displayable characters is of less importance to an SPFLite user than it would be to an IBM ISPF user.

The `P' . '` Picture code is provided for ISPF compatibility, but you may not have extensive need for it. **If you chose to**, you could probably define the Normal Characters string so that a `P' . '` Picture would detect essentially the same unprintable characters that it would on the mainframe. However, it would be very unusual to try to emulate a mainframe 3270 in this manner.

Specifying a Regular Expression

Contents of Article

[Introduction](#)

[Performance Considerations](#)

[Compatibility Considerations](#)

[Examples](#)

[Using the SPFTest Program](#)

Introduction

Using a Regular Expression string in a **FIND**, **CHANGE**, **JOIN**, **EXCLUDE** or similar command provides you with a powerful tool for searching. If you have never used regular expressions they may appear quite complex. However many users will already be familiar with them from their use on other systems like Linux or Unix.

Note: Users familiar with regular expressions should be aware that there are many variants of regular expression syntax implemented in different systems and applications, some with fairly subtle differences. The description below describes the regular expression engine supported by SPFLite. The RegEx engine chosen for SPFLite is PCRE (Perl Compatible Regular Expression). This engine is widely used and supports a robust RegEx syntax. Because regular expressions can be complex, SPFLite provides a small program to allow you to experiment with Regular Expressions and become familiar with their use. See [Using the SPFTest Program](#) for details.

There are several sites on the web providing assistance in learning about and testing Regular Expressions. One such tutorial can be found at www.regexlab.com.

The regular expression must be enclosed in quotes (single, double, or accent) preceded or followed by the character **R**.

Examples:

`R 'abc$'`
`'abc$' R`

SubTopics

- [PCRE Regular Expression Syntax Summary](#)
- [Quoting](#)
- [Characters](#)
- [Character Types](#)
- [General category Properties for \p and \P](#)
- [PCRE Special Category Properties for \p and \P](#)
- [Character Classes](#)
- [Quantifiers](#)
- [Anchors and Simple Assertions](#)
- [Match Point Reset](#)
- [Alternation](#)
- [Capturing](#)
- [Atomic Groups](#)
- [Comments](#)
- [Option Setting](#)
- [Newline Convention](#)
- [What \R Matches](#)

- [Lookahead and Lookbehind Assertions](#)
- [Backreferences](#)
- [Subroutine References \(Possibly Recursive\)](#)
- [Conditional Patterns](#)
- [Backtracking Control](#)
- [Callouts](#)

PCRE Regular Expression Syntax Summary

This PCRE Syntax documentation courtesy of www.pcre.org - Copyright © 1997-2014 University of Cambridge.
 This is a summary only. The full description can be found at
<http://www.pcre.org/original/doc/html/pcrepattern.html>

QUOTING

```
\x      where x is non-alphanumeric is a literal x
\Q...E  treat enclosed characters as literal
```

Note that in PCRE, the code `\Q` differs from prior SPFLite RegEx usage of `\q`.
 See [Compatibility Considerations](#) below for more information.

CHARACTERS

```
\a      alarm, that is, the BEL character (hex 07)
\cx    "control-x", where x is any ASCII character
\e      escape (hex 1B)
\f      form feed (hex 0C)
\n      newline (hex 0A)
\r      carriage return (hex 0D)
\t      tab (hex 09)
\0dd    character with octal code 0dd
\ddd    character with octal code ddd, or backreference
\o{ddd..} character with octal code ddd..
\xhh    character with hex code hh
\x{hhh..} character with hex code hhh..
```

Note that `\0dd` is always an octal code, and that `\8` and `\9` are the literal characters "8" and "9".
 Note that in PCRE, the code `\c` differs from prior SPFLite RegEx usage.
 See [Compatibility Considerations](#) below for more information.

CHARACTER TYPES

These codes represent any character except for newline. In "dotall mode", these codes represent any character whatsoever.

\C	one data unit, even in UTF mode (best avoided)
\d	a decimal digit
\D	a character that is not a decimal digit
\h	a horizontal white space character
\H	a character that is not a horizontal white space character
\N	a character that is not a newline
\p{xx}	a character with the xx property
\P{xx}	a character without the xx property
\R	a newline sequence
\s	a white space character
\S	a character that is not a white space character
\v	a vertical white space character
\V	a character that is not a vertical white space character
\w	a "word" character
\W	a "non-word" character
\x	a Unicode extended grapheme cluster

By default, `\d`, `\s`, and `\w` match only ASCII characters.

Note that in PCRE, the code `\s` differs from prior SPFLite RegEx usage.

See [Compatibility Considerations](#) below for more information.

GENERAL CATEGORY PROPERTIES FOR \p and \P

C	Other
Cc	Control
Cf	Format
Cn	Unassigned
Co	Private use
Cs	Surrogate
L	Letter
Ll	Lower case letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter
Lu	Upper case letter
L&	Ll, Lu, or Lt
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
Nl	Letter number
No	Other number
P	Punctuation
Pc	Connector punctuation
Pd	Dash punctuation
Pe	Close punctuation
Pf	Final punctuation
Pi	Initial punctuation
Po	Other punctuation
Ps	Open punctuation
S	Symbol
Sc	Currency symbol
Sk	Modifier symbol
Sm	Mathematical symbol
So	Other symbol
Z	Separator
Zl	Line separator
Zp	Paragraph separator
Zs	Space separator

PCRE SPECIAL CATEGORY PROPERTIES FOR \p and \P

Xan	Alphanumeric: union of properties L and N
Xps	POSIX space: property Z or tab, NL, VT, FF, CR
Xsp	Perl space: property Z or tab, NL, VT, FF, CR
Xuc	Universally-named character: one that can be represented by a Universal Character Name
Xwd	Perl word: property Xan or underscore

CHARACTER CLASSES

[...] positive character class
 [^...] negative character class
 [x-y] range (can be used for hex characters)
 [:xxx:] positive POSIX named set; see list below
 [:^xxx:] negative POSIX named set; see list below

xxx can be one of the following:

alnum	alphanumeric
alpha	alphabetic
ascii	characters in the range of 0 to 127 (\x00 to \x7F)
blank	space or tab
cntrl	control character
digit	decimal digit
graph	printing, excluding space
lower	lower case letter
print	printing, including space
punct	printing, excluding alphanumeric
space	white space
upper	upper case letter
word	same as \w
xdigit	hexadecimal digit

QUANTIFIERS

```

?          0 or 1, greedy
?+         0 or 1, possessive
??         0 or 1, lazy
*          0 or more, greedy
*+         0 or more, possessive
*?         0 or more, lazy
+          1 or more, greedy
++         1 or more, possessive
+?         1 or more, lazy
{n}        exactly n
{n,m}      at least n, no more than m, greedy
{n,m}+     at least n, no more than m, possessive
{n,m}?     at least n, no more than m, lazy
{n,}        n or more, greedy
{n,}+      n or more, possessive
{n,}?      n or more, lazy

```

Note that in PCRE, the quantifier code `?` may be used in some cases like the code `\s` from prior SPFLite RegEx usage.

See [Compatibility Considerations](#) below for more information.

ANCHORS AND SIMPLE ASSERTIONS

```

\b          word boundary
\B          not a word boundary
^          start of subject
            also after internal newline in multiline mode
\A          start of subject
$          end of subject
            also before newline at end of subject
            also before internal newline in multiline mode
\z          end of subject
            also before newline at end of subject
\z          end of subject
\G          first matching position in subject

```

MATCH POINT RESET

```
\K          reset start of match
```

`\K` is honored in positive assertions, but ignored in negative ones.

ALTERNATION

```
expr|expr|expr...
```

CAPTURING

(...)	capturing group
(?<name>...)	named capturing group (Perl)
(?'name'...)	named capturing group (Perl)
(?P<name>...)	named capturing group (Python)
(?:...)	non-capturing group
(? ...)	non-capturing group; reset group numbers for capturing groups in each alternative

ATOMIC GROUPS

(?>...)	atomic, non-capturing group
---------	------------------------------------

COMMENT

(?#...)	comment (not nestable)
---------	-------------------------------

OPTION SETTING

(?i)	caseless
(?J)	allow duplicate names
(?m)	multiline
(?s)	single line (dotall)
(?U)	default ungreedy (lazy)
(?x)	extended (ignore white space)
(?-...)	unset option(s)

The following are recognized only at the very start of a pattern or after one of the newline or \R options with similar syntax. More than one of them may appear.

(*LIMIT_MATCH=d)	set the match limit to d (decimal number)
(*LIMIT_RECURSION=d)	set the recursion limit to d (decimal number)
(*NO_AUTO_POSSESS)	no auto-possessification (PCRE_NO_AUTO_POSSESS)
(*NO_START_OPT)	no start-match optimization (PCRE_NO_START_OPTIMIZE)
(*UTF8)	set UTF-8 mode: 8-bit library (PCRE_UTF8)
(*UTF16)	set UTF-16 mode: 16-bit library (PCRE_UTF16)
(*UTF32)	set UTF-32 mode: 32-bit library (PCRE_UTF32)
(*UTF)	set appropriate UTF mode for the library in use
(*UCP)	set PCRE_UCP (use Unicode properties for \d etc)

Note that LIMIT_MATCH and LIMIT_RECURSION can only reduce the value of the limits set by the caller of `pcre_exec()`, not increase them.

NEWLINE CONVENTION

These are recognized only at the very start of the pattern or after option settings with a similar syntax.

(*CR)	carriage return only
(*LF)	linefeed only
(*CRLF)	carriage return followed by linefeed
(*ANYCRLF)	all three of the above
(*ANY)	any Unicode newline sequence

WHAT \R MATCHES

These are recognized only at the very start of the pattern or after option setting with a similar syntax.

(*BSR_ANYCRLF)	CR, LF, or CRLF
(*BSR_UNICODE)	any Unicode newline sequence

Note: Be careful not to confuse `\R` with `\r`. The code `\r` only means a carriage return character (`\x0D`), while the meaning of `\R` depends on the settings above.

LOOKAHEAD AND LOOKBEHIND ASSERTIONS

(?=...)	positive look ahead
(?!...)	negative look ahead
(?<=...)	positive look behind
(?<!...)	negative look behind

Each top-level branch of a look behind must be of a fixed length.

BACKREFERENCES

\n	reference by number (can be ambiguous)
\gn	reference by number
\g{n}	reference by number
\g{-n}	relative reference by number
\k<name>	reference by name (Perl)
\k'<name'>	reference by name (Perl)
\g<name>	reference by name (Perl)
\k<name>	reference by name (.NET)
(?P=name)	reference by name (Python)

SUBROUTINE REFERENCES (POSSIBLY RECURSIVE)

```

(?R)           recurse whole pattern
(?n)           call subpattern by absolute number
(?+n)          call subpattern by relative number
(?-n)          call subpattern by relative number
(?&name)       call subpattern by name (Perl)
(?P>name)      call subpattern by name (Python)
\g<name>      call subpattern by name (Oniguruma)
\g'name'       call subpattern by name (Oniguruma)
\g<n>          call subpattern by absolute number (Oniguruma)
\g'n'          call subpattern by absolute number (Oniguruma)
\g<+n>         call subpattern by relative number (PCRE extension)
\g'+n'         call subpattern by relative number (PCRE extension)
\g<-n>         call subpattern by relative number (PCRE extension)
\g'-n'         call subpattern by relative number (PCRE extension)

```

CONDITIONAL PATTERNS

```

(?:condition)yes-pattern
(?:condition)yes-pattern|no-pattern

(?:n) ... absolute reference condition
(?:+n) ... relative reference condition
(?:-n) ... relative reference condition
(?:<name>) ... named reference condition (Perl)
(?:'name') ... named reference condition (Perl)
(?:name) ... named reference condition (PCRE)
(?:R) ... overall recursion condition
(?:Rn) ... specific group recursion condition
(?:R&name) ... specific recursion condition
(?:DEFINE) ... define subpattern for reference
(?:assert) ... assertion condition

```

BACKTRACKING CONTROL

The following act immediately they are reached:

```

(*ACCEPT)      force successful match
(*FAIL)        force backtrack; synonym (*F)
(*MARK:NAME)   set name to be passed back; synonym (*:NAME)

```

The following act only when a subsequent match failure causes a backtrack to reach them. They all force a match failure, but they differ in what happens afterwards. Those that advance the start-of-match point do so only if the pattern is not anchored.

(*COMMIT)	overall failure, no advance of starting point
(*PRUNE)	advance to next starting character
(*PRUNE:NAME)	equivalent to (*MARK:NAME) (*PRUNE)
(*SKIP)	advance to current matching position
(*SKIP:NAME)	advance to position corresponding to an earlier (*MARK:NAME); if not found, the (*SKIP) is ignored
(*THEN)	local failure, backtrack to next alternation
(*THEN:NAME)	equivalent to (*MARK:NAME) (*THEN)

CALLOUTS

(?C)	callout
(?Cn)	callout with data n

Performance Considerations

Be aware that the *, + and ? meta-characters can cause the regular expression engine to perform backtracking, which can cause its pattern-matching to run a little slower. As with all regular expressions, performance is dependent on file size and the complexity of the expression.

Compatibility Considerations

The PCRE regular expression syntax is mostly compatible with what SPFLite offered in prior releases. You will generally not need to be concerned about compatibility if you only issue RegEx expressions on the command line with R strings. If you have stored regular expressions, either in a macro or a keyboard definition, you should be aware of the following differences.

Pre-PCRE Regular Expression syntax prior to release 8.3	PCRE Regular Expression syntax as of release 8.3
\c Signifies a case-sensitive search	PRCE uses this code in an incompatible way. The PCRE code \cx to indicate a "control" character. For example, if you want to use a Ctrl-A as a literal value, you would specify \cA. SPFLite honors the CASE C/T setting when executing a PCRE regular expression. Where you might previously have used \c to mean "case sensitive", you can now issue a CASE C primary command prior to executing a regular expression. If you want to embed a code in your RegEx string to control case sensitivity, (?i) means case insensitive (like CASE T) and (?-i) means case sensitive (like CASE C).
\q Signifies a literal double-quote character "	PCRE does not implement the \q code. It does have a \Q code, but with an unrelated meaning. The prior RegEx engine allowed this code to make

	<p>it easier to specify literal double-quote characters in Basic strings. You will generally not need to do this. A double-quote character " has no special meaning in a PCRE RegEx, and can be directly specified as a data value, as long it's not also being used to enclose the R string as a whole.</p> <p>If you are creating a RegEx string within a SPFLite macro script, you can use the Basic string equate \$DQ to get a double-quote character.</p> <p>You can also insert a double quote by using the RegEx hex value for the ASCII double quote, which is \x22.</p>
<p>\s</p> <p>Signifies a request to match the shortest-possible string</p>	<p>PRCE uses this code in an incompatible way. The PCRE code \s is used to search for a "white space" character.</p> <p>PCRE uses the term "greedy" when a string-matching process could match against a variable number of characters, and the longest possible length is taken. When the shortest-possible length is taken, this is called "lazy" or "non-greedy". Thus, the old code of \s meant "lazy" or "non-greedy" in PCRE terminology.</p> <p>To set "lazy" matching in PCRE, you can use a general option setting code of (?U), and greedy mode would be (?-U). These options affect the entire RegEx expression. PCRE matching is "greedy" by default, so the code (?-U) is not usually required.</p> <p>PCRE also lets you control the greedy-vs-lazy matching on a finer level, using a ? in certain types of syntax. See the PCRE documentation under QUANTIFIERS for more information.</p>

Examples

Given a file with lines:

```
"Blah blah blah blah"
"Please send email to FredFarkle@mydomain.com"
"Thank you very much."
```

The line with the email address could be found with:

```
FIND R" ([a-zA-Z0-9._/+-]+) (@[a-zA-Z0-9.-]+) "
```

Given the lines:

"Balance due by 15th of the month"

"Amount owed: \$42.75 and is overdue!"

The line with the Balance amount could be found with:

FIND R"\\$[0-9.,]+"

Working with Mapping Strings

Documentation on Mapping Strings is divided into two major sections.

The [Quick Reference Guide](#) provides concise answers to simple questions, and is well indexed. If you need mapping-command syntax or examples, this is where you should look.

For in-depth coverage on the topic of Mapping Strings, refer to the [Mapping String Comprehensive Guide](#). It contains complete coverage of this topic, which you should read once to understand the underlying concepts. After that, the Quick Reference will be sufficient in most cases.

Mapping String Comprehensive Guide

SPFLite introduces a new editing concept: The use of “mapping strings” to reorder, reformat and transform characters within strings that are found by the **CHANGE** command. Mapping strings are defined by a new string type code of **M**.

You will find that mapping strings have an wide array of features – probably **far more** than you will ever need at any given time. It is much like a “Swiss Army Knife” – a tool having a lot of parts, some of which you will use and some you won’t. Don’t let that hold you back from trying them. You don’t need to read everything. Just focus on the functions you are interested in. If all you want to do is rearrange columns, you don’t need to know how to convert data from ASCII to hex or reformat a decimal value. Let your own editing requirements drive which areas you want to learn about and explore.

And **yes**, this Working With article is quite long. Just take it one section at a time. If you have some simple task to do, you may save time by going to the [Quick Reference Guide](#) and look at the [General Examples](#) section. You may find something there to immediately get you going.

For some tasks, mapping strings may be powerful enough that you could use them instead of writing an SPFLite edit macro. If your task involving reformatting data, chances are good that you may be able to do some or all of it with a mapping string. The main thing you should remember is that the features are here and available, even if you have to reread the directions each time you them to use. Like everything else in SPFLite, you will remember the parts you use the most often, and look up the rest.

Because mapping strings are a feature not present in IBM ISPF, you are encouraged to fully read through this article at least once, to familiarize yourself with all the new capabilities they have to offer and gain the most benefit from them. For the same reason, we recommend that you test and experiment with mapping strings, perhaps using a small file of test data, to be sure you understand how these features work and to verify you have specified your mapping string correctly before applying it to real data files, until you gain experience with it. To help with the learning curve, there is a test program you can run to watch how these commands work without having to set up test files. Refer to the section [Using the mapping test program](#) at the end of this article for more information.

Finally, if any issues or questions arise, the SPFLite forum is available to assist you.

Preface

Note: This article is the **long** explanation of mapping strings, when you need the full background on this concept and “want to read a book about it”. If you **don’t** want to read a book about it, go to [Quick Reference Guide](#) for concise information on specific commands and usage. It contains links to each command to help get you immediately “up and running” if you just need a quick answer on some point.

It is recommended (and you will benefit) if you **do** read through this article at least once. After you understand the concepts, the Quick Reference should be enough to complete most tasks you may have.

As a new technology, mapping strings should be considered as an experimental feature. That does not mean it doesn’t work, or has known issues, but there could be *unknown* issues that won’t be apparent until you as the SPFLite user community gain experience with it, and (hopefully) provide us with feedback on how it’s working for you.

You will find you can accomplish a great deal with mapping strings, but no single technique will do everything. Mapping strings are just one feature in an array of techniques you can use to automate your editing tasks and get your work done faster, each of which having its own advantages. These other techniques include:

- [Execute-macro strings](#), also known as “E strings” or “E macros”
- Traditional SPFLite macros, used as primary commands or as line commands
- SET and SET ALIAS commands to abbreviate long primary command sequences
- Command chaining
- Excluded lines, User lines and Tags
- Multi-Edit sessions
- Power Typing
- Keyboard macros and keyboard functions

For the more adventurous among you, it is possible to write an SPFLite macro that incorporates **CHANGE** commands with **M** strings inside of calls to **SPF_Cmd()**. However, you cannot write an SPFLite macro that incorporates **CHANGE** commands with **E** strings inside of calls to **SPF_Cmd()**, since that would result in a nested macro execution environment, which is not supported.

Because mapping strings and **E** strings are new technologies, it must be admitted that all possible combinations of these technologies with all previously existing SPFLite features have not been tested. They **should** work, but if you run into issues, please let know so we can look into it.

Fundamental Concepts

Mapping strings allow for *data reordering*, *data reformatting*, and *data transformation*.

Data reordering is a type of **CHANGE** operation in which the relative character positions of string are altered. For example, changing a string "ABCD" into "DCDA" is a simple data reordering operation. *Data reordering* may also include *data extraction*, because it is not necessary for *all* of the original data to be included in the resulting string value.

Data reformatting is a type of **CHANGE** operation in which additional characters are added to the original data, or where the original data is modified by having certain common text operations applied to it. For example, modifying a string "ABCD" into "*** AB-CD ***" is a data reformatting operation. Refer to the [Quick Reference Guide](#) to see [examples](#) in the how these and other operations can be performed with mapping strings.

Data transformation is a type of **CHANGE** operation in which the representation of characters in the original data is fundamentally changed. Commands are available in a mapping string to change data from character-mode to hex-mode using either ASCII or EBCDIC character sets, and to alter data by the use of a translation table. That is, it's possible to replace all instances of A with 1, B with 2 and C with 3 – regardless of where the A, B and C appeared in your data – in a single operation. It is also possible to mutually exchange a pair of strings in your data, such as changing each occurrence of the word "one" into "two" and vice-versa. Without the data transformation capabilities in mapping strings, these actions could require up to 3 passes over your data, and the effort could incorrectly change some data, making that effort less reliable. In contrast, a mapping-string data transformation alters all of the effected data at the same time. You can also transform certain numeric data by converting it between hex and decimal and/or adjusting its length and formatting.

A **mapping string** is a string value appearing on a **CHANGE** command. It takes the place of a conventional *change string* in a **CHANGE** command. (Some documentation may describe the *change string* of a **CHANGE** command as *string-2*, whereas the *find string* part of the **CHANGE** command may be called *string-1*.) SPFLite uses mapping strings on **CHANGE** commands to define how a given reordering, reformatting or transformation operation takes place. Reordering, reformatting and transformation operations can be applied at the same time. (When you use these strings just for reformatting or transformation, rather than for reordering, you might think of these strings as *modification strings* rather than *mapping strings*. However, we use the term *mapping string* to cover all uses, since you are free to do several desired actions at the same time.)

Mapping strings are introduced by a quoted string value with a string type code of **M**. As with other SPFLite string types, **M** strings may be enclosed in any of the three types of quotes: (') single, (") double, or (`) accent), the **M** type code is case-insensitive, and the letter **M** may appear either at the beginning or at the end of the string, as **M'xxx'** or **'xxx'M**. Because an **M** string may contain mapping items with their own quoted string operands, you may find it convenient to enclose the **M** string itself in " double quotes, and use ' single quotes or ` accent quotes inside the **M** string, for clarity and ease of typing.

An **M** string is only allowed on **CHANGE** commands, and only as the second string operand. (You cannot attempt to *find* strings of type **M**.)

Why would you use mapping strings? The main reason is to convert or reformat existing structured data from one pattern to another. Example data might include part numbers, serial numbers, telephone numbers, record keys, etc. Mapping strings can be used on any sort of file, though the types of reformatting you would perform on data files vs. program source code files is likely to be different.

When you use a mapping string on a **CHANGE** command, the first string operand for that command can be any valid SPFLite *find-string* type. However, if you had a *find string* like '**ABCD**', there would be no real need to

reformat it as a literal 'DCBA' using *mapping*, since you could already do that with a simple command of **CHANGE 'ABCD' 'DCBA'**. The place where mapping is the most useful is when the *find-string* locates a variety of different data values. In practice, that means a **CHANGE** command that has a mapping string will normally have a *find-string* which is of type Picture (**P**) or a Regular Expression (**R**), rather than a literal string of type **C**, **T** or **X** or unquoted text.

Note: It is possible to apply a mapping string operation to an **entire line of data**, by finding whole lines at a time with the *find string* of your **CHANGE** command, using a Regular Expression of **R'^.*\$'**. Be aware that if you do this, the *source string* value of various lines will likely be of differing sizes. You would need to consider how you would refer to particular columns of varying-length data in such circumstances; the use of *end-relative column notation* could prove helpful.

Note: Starting with release 8.3 of SPFLite, a Regular Expression of **R'^.*\$'** can find lines of length zero, because the Regular Expression engine has changed for 8.3; that did not occur in prior releases. This should not affect most users, but if you should encounter any issues because of this, be sure to let us know on the SPFLite forum, so we can make sure everything is working properly.

Basic features of data mapping

Mapping is done in the course of executing a **CHANGE** command.

The *find string* in the **CHANGE** command (sometimes called *string-1*) can be any valid string type, such as unquoted strings or **C**, **T**, **X**, **P**, or **R** strings. As noted above, for mapping, you will normally use *find strings* of type **P** or **R**.

Note: Format (**F**) strings are not allowed as *find strings*.

The data you find with your *find string* is called the *source string*, since it is the original source data you will be processing in your mapping operation.

Once that string is found, the characters of the *source string* are *enumerated* from left to right, with the left-most enumerated character position being 1, and the right-most being the length of the found string.

Note: Strings of length zero are never found by the standard SPFLite search engine when searching for a **Picture** string, because there is literally no data to find. However, starting with release 8.3 of SPFLite, a Regular Expression **can** find strings of length zero, because the Regular Expression engine has changed for 8.3; that did not occur in prior releases. This should not affect most users, but if you should encounter any issues because of this, be sure to let us know on the SPFLite forum, so we can make sure everything is working properly

Mapping is done by selecting one or more character positions from the *source string*, and applying a mapping item to it. A *mapping item* is either a simple column reference to the *source string*, or a *mapping command* which itself can use a column reference as an operand to its processing.

For example, if we want to change a string like **ABCD** into **DCBA**, we first have to find strings like **ABCD**. Recall from the discussion above, we are **not** interested in finding an actual value of **ABCD**, because all that requires is simply a find string of **ABCD**. What we are interested in is finding strings *like* **ABCD** – that is, word values containing 4 letters.

Suppose these strings were found as the result of a **WORD** picture of **P'@@@@'**. The *enumeration* of the character positions of a string *like* **ABCD** is:

A = 1, **B** = 2, **C** = 3, **D** = 4

To reorder these character values, we have to select them in the desired new order. That is, to get a string *like* **DCBA**, we have to select the character in position **4** of the *source string* value, then character **3**, then **2**, then **1**. That is what a mapping string does. It defines that new ordering. So, to get a string *like* **DCBA**, it's necessary to select, and then map, these characters into the order of **4, 3, 2, 1**.

A complete SPFLite command to do what is described above would look like this:

```
CHANGE P'@@@@' WORD M'4-1'
```

Mapping strings allow for these and many additional operations, including:

- altering the case of the *result string*

- defining ranges and reverse ranges to select columns of data
- inserting text
- deleting text
- repeating text
- accounting for otherwise-unselected source data
- setting the default CASE C/T processing
- aligning text left, right, or center with padding or truncation to a given length
- trimming of leading or trailing characters
- padding with leading or trailing characters
- suppressing of leading zeros
- converting character data based on a translation table
- converting character data between display and hex in ASCII or EBCDIC
- converting values between decimal and hex with reformatting
- exchange two different strings
- moving data between locations within the *result string*

Understanding how string mapping works

When you request a string to be **mapped**, you are taking data from a *source string* and using it to create a *result string*. The *mapping string* can be thought of as a list of ‘commands’ or ‘instructions’, which are operated upon in left-to-right order. These instructions, or *mapping items*, consist of column ranges, command codes, and optional string and numeric operands, and are described below.

Visually, think of it like this:

“source string” → { mapping items … } → “result string”

When the mapping operation begins, SPFLite will initially set the *result string* to a ‘null’ (zero-length) value. The *result string* is then subsequently changed as a result of the mapping commands you specify.

You may reference the *source string* using as many commands as you wish. As you do, each successive command will update the contents of the *result string*. However, **the source string itself is never changed, and may be thought of as a constant, read-only value.**

Just to clarify this, the *source string* is essentially an “input parameter” to a “mapping function”. The mapping function does **not** change this input parameter during the course of its operation while interpreting your mapping commands. That means you can refer to the *source string* multiple times with multiple mapping items, and the contents of the *source string* always retains the original string found by the **CHANGE** command. Once you have created your *result string* by specifying one or more mapping items, that *result string* replaces the found string in your data file.

For example, in the command noted above, **CHANGE P'@@@@' WORD M'4-1'**, if the **CHANGE** command finds the string ABCD, that string is the *source string* for the duration of whatever mapping commands you issue within your M string. The *source string* never changes from being ABCD, since that is an “input parameter” to the mapping process. The “output parameter” from the mapping process is your *result string*. That output parameter is then sent back to the **CHANGE** command processor to complete the **CHANGE** process, by replacing your original data of ABCD with DCBA.

Each column-reference selects and **copies** data from the *source string* and places it in the *result string*, successively appending the selected values on the right side of it, to produce an updated version of the *result string*. These are known as **copying commands**. Each *mapping item* generally causes the *result string* to built up and get longer and longer until the final result is produced. (A few operations are an exception to this, because they may make the string shorter or update it in place.)

Some operations such as case alteration, alignment or trimming, operate on and **modify** the *result string* as a whole, as *it exists as of the point where the operation in the mapping string is reached*. These are known as **modifying commands**. Thus, it is possible, for example, to “trim” the *result string* by a trimming command in the middle of the mapping string, and then go on to make the *result string* longer with additional mapping items after that.

Mapping items which are **modifying commands** do not use or reference the contents of the *source string*, but **only** the *result string*. For example, suppose you had a *source string* of “abcd”, and applied a mapping string to it of **M'1-2 UC 3-4'**. This would work as follows:

- Columns 1-2 of the *source string* are selected as “ab”. This value is **copied** to the *result string*, **which is initially empty**. The *result string* now contains “ab”. (Technically, the string “ab” is appended to a null result string, resulting in “ab”, but the effect is the same as if it were simply copied.)

- The case alteration command **UC** is applied to **modify** the current *result string*. The *result string* now contains “**AB**”
- Columns 3-4 of the *source string* are selected, “**cd**”. This is **copied** and appended to the end of the *result string*, which at this point had “**AB**” because of the prior **UC** command. The final value of the *result string* is now “**ABcd**”.

In some cases, if you specify a column-range on a command code that is outside the range of data columns or your *source string*, or you have otherwise specified your command improperly, no change will be performed on your data. If you issue a **CHANGE** command with an **M** string, but nothing seems to be happening, check your data and your **M** string to see if you have specified it correctly. Also, check to see if your **CHANGE** command has produced a mapping-string error message, which may help you to identify and correct the problem.

If you have a legitimate reason to attempt to access column positions that don't (yet) exist in your *source string*, the best way to achieve this may be to first pad the data to the desired length (likely with spaces), and then proceed with any additional modifications to your data that you need. You cannot pad the *source string* itself, but you can pad a copy of it in the *result string*.

Understanding dot notation

Understanding dot notation

All command and features in mapping strings revolve around a basic concept, which is providing the ability to inspect and modify character data. The data that is being operated on at any given point may originate either from the *source string* or the *result string*. When a command can reference and copy data directly from the *source string*, it is a **copying command**. When it can only refer to data already in the *result string*, it is a **modifying command**.

To keep the various command codes as simple as possible, the design has considered which **data origin** a given command is likely to need to reference the most often, and that data origin is the *mapping commands default origin*. For example the case alteration command **UC** is applied by default to the *result string*, and so the **UC** command has the *result string* as its *default data origin*.

In addition to commands which operate on the *result string* by default, mapping commands that take a column-range operand can use the `.` dot symbol to indicate that a reference to the current contents of the *result string* is being made. This is known as **dot notation**.

If a `.` dot symbol precedes a column reference, like `.1-4`, that column reference is used to select a substring from the current contents of the *result string*. You can use *dot notation* anywhere a regular column reference is allowed, either by itself, or as part of a column-range operand on a particular command code.

Visually, think of *dot notation* like this:

“result string” → { *copying commands with dot notation* ... }
→ *“revised result string”*

You may not often need to use this technique, but it is available if you do. The main reason for dot notation is to allow a **copying command** to be used as a **modifying command** instead. For example, the command **AX** is a **copying command**. Thus:

1. A mapping item of **AX1-4** would convert the first 4 characters of the *source string* into 8 hex-digit characters and place them in the *result string*
2. A mapping item of **AX1+** would convert the entire *source string* into hex-digit characters, producing a string twice as long and place them in the *result string*.
3. Because of the **auto-reference** feature, if a simple **AX** command is issued with no column numbers after it, it implies that the complete *source string* is referenced by the **AX** command, and the value it produces is placed into the *result string*. So, if the *source string* contained “abcd” and you issued this **AX** command as the first thing in the mapping string, it would convert those 4 characters, then place the converted into the *result string*, producing “61626364”.
4. A mapping item of **AX.1-4** would convert the first 4 characters of the *result string* into 8 hex-digit characters. Those 8 characters would be appended to the end of the *result string*. So, if the *result string* already contained “abcd” and you issued this command, it would convert those 4 characters, then place the converted result after those 4 characters that were already there, producing “abcd61626364”.
5. A mapping item of **AX.** would convert the entire *result string* into hex-digit characters, producing a string

three times as long (the original data put 2 hex digits for each original character); those characters would be appended to the end of the *result string*. Note that **AX.1+** and **AX.** so happen to produce the same result, since a lone . dot references the entire *result string*, and it implies **AX.1+**

6. If you specify *dot notation* to force a **copying command** to modify *result string* data, the dot notation will cause the *auto copy* feature normally associated with **modifying commands** to take place.

For example, if your *source string* contains "abcd" and you issue a mapping command of **AX.1-4**, and if this were your only mapping command, the dot notation would cause the *source string* to be initially copied to the *result string* as "abcd" and then columns **1-4** of the *result string* (in this case, the entire value) would have the **AX** command applied to it, and the value it produces is placed at the end of the "abcd" in the *result string*, giving a revised result string value of "**abcd61626364**".

Usually, you wouldn't want both the original value of "abcd" and the results of the **AX** command joined together like this. In this example, the correct way to do this would be to note that, because **AX** is already a **copying command**, you can convert the *source string* directly simply by using **AX** alone without the **.1-4** part, giving a final value of "**61626364**". The best way to understand this is to experiment using various commands on some test data, and observe how they operate. You can also use the Mapping Test Program for learning and experimentation. The Mapping Test Program is discussed at the end of this article.

When you read the descriptions below, pay close attention as to whether the mapping item involves the *source string*, the *result string*, or both. Look for the syntax description of the *column-reference* operand under each command's documentation in the Quick Reference. You will find under each command a note as to whether the command is a **copying command** or a **modifying command**. A few specialized commands are neither copying nor modifying commands.

How a mapping string is specified

Mapping strings are defined with a string type code of **M**. As with all SPFLite strings, this code may appear at the beginning or at the end of the string, as **M'xxx'** or **'xxx'M**. The **M** is case-insensitive. Inside this string, blanks outside of inserted-text literals are ignored other than as delimiters. However, the **M** string cannot be entirely blank or zero-length; this would imply that the string was to perform no mapping operation at all. An empty **M** string is reported as an error.

A mapping string must have at least one valid command code or column reference (even if that command does not happen to alter the *source string*). It is not an error to specify a valid mapping string that does not happen to result in a data change to the contents of your *source string*. In addition, any time you issue a mapping string that contains a syntax error or other problems, your original source data will not be changed.

A mapping string is a quoted string value containing one or more *mapping items*. *Mapping items* must be delimited from each other by one or more blanks. An *individual* mapping item, including any optional command codes, string operand and/or numeric operands, must not have any embedded spaces or other characters not shown as part of the syntax, unless those are within a quoted string operand.

When an item references character positions that don't exist in the particular found string, they effectively map to null (zero length) character values. For example, if you try to use a mapping string to select column 5 from a data value, but the value is only of length 4, it is not an error, but no value is selected for the absent column. The request to do *that part of the mapping* is simply ignored. The way this is handled is of particular importance when using a Regular Expression (a type-R find-string) to locate data, because a Regular Expression may be used to locate strings of varying lengths. See the Examples in the [Quick Reference](#) for more information.

When certain kinds of items are preceded by a \ backslash, it reverses the order of the *result string*. The \ backslash must be immediately followed by a number, but must itself occur at the beginning of the mapping string or be preceded by a delimiter. (That is, you cannot run two numbers together with a backslash between, like "12\45+". This must instead appear as "12 \45+" to make your intentions clear.) See the discussion below for details.

Column-range operands can accept a special / or \ punctuation as part of [mixed-mode end-relative notation](#), discussed next.

Guidelines for writing mapping strings

Much like Regular Expressions, mapping strings, with their various commands and operands, can be somewhat cryptic at times, and for all but the simplest cases they may not be something that will easily “roll off the tip of your tongue”. Naturally, the more complex your mapping or modification operations are, the more complex the mapping string will be. You may need to reread the documentation (perhaps each time you use it), and plan out in advance what these strings will contain.

You may benefit by saving off some mapping commands you have found useful, along with a few descriptive notes, into a text file to have them handy, like having a “cheat sheet”. That way, if you need to use them again, you can refer to your notes instead of always having to reread this documentation. Or, you can specifying an external mapping string item of **SV** to fetch a **SET** variable containing part of a mapping string. You can also use an SPFLite **SET** variable to define an entire mapping string, and then you would specify it on your **CHANGE** command with an **=** equal sign followed by the name of the **SET** variable.

You can look at the provided examples to help understand how mapping strings are written; if one is close to what you need, you may be able to copy and tailor it. When working with your own data, there are a few ways to go about defining a mapping string. For short data, the simplest way may be to just write some sample data on a piece of paper, drawing lines where various substrings of your data are located, and counting the columns. Then, write down each ‘piece of your data as you want it reformatted, in left-to-right order, so that each ‘piece is an *item* in a mapping string, as described above.

For longer data, you can use SPFLite to help. If you highlight a string on a line, you will see its length displayed on the status line, like **Len 8**. If you place your data on a line by itself, left-justified on column 1, you will see the length of the line on the status line when the cursor is on that line of data, like **Line Len 8**. Be certain the line is *trimmed* to sure you don’t count any trailing blanks. You can trim a line of data by using the **TR** line command. The *virtual highlighting pens* feature of SPFLite may also be useful.

You also have the **L/C** display on the status line, which shows the Line number and Column number where the cursor is located. If your data is on line 125, you will see a display of **L 000125 C 8** when the cursor is on the last column your data.

Finally, you can issue the **COLS** line command to show you column positions.

Bear in mind that the “column numbers” or “column ranges” in a mapping string are the **relative positions** of your data. Those numbers have nothing to do with where (what actual column positions) on a *data line* that your data is found in the course of executing a **CHANGE** command. You must visualize your data as if it were left-justified on column 1 of a line, for purposes of numbering the positions, even though your data may actually appear anywhere, in any columns of a line.

Remember that many mapping commands are what is termed “modifying commands”. That means they are designed to alter the contents of the *result string*. To use such commands on data that is in the *source string*, you have to first copy some or all of the *source string* to the *result string*. The easiest way to do that is with a column-copying command of **1+** at the beginning of your mapping string. You will see examples in the documentation that use this technique. However, for many commands (but not all), the *auto-reference* and *auto-copy* features mean that oftentimes you won’t need to use a *column reference* of **1+** to reference or copy the entire *source string*, because it will be implied and will happen automatically.

You may come across cases where a single mapping operation is not powerful enough to do what you need. When that happens, it’s possible you could do part of the mapping in one pass, and in the process, insert or

append to the *result string* some special character of your choosing as a “marker”. Then, you would go back with another **CHANGE** command that looked for strings containing that special “marker” character, and do the remaining data reformatting that you were unable to complete in the first pass – either with another **M** string or perhaps a conventional string of type **C**, **T** or **X**.

To improve performance, and limit the second pass to only those lines affected by the first pass, you could use *Command Chaining* techniques involving a “**Z tag**” of :ZF on your second **CHANGE** command. See the Help article [Working with Command Chaining](#) for more information.

When using this technique, be mindful of what characters have special meaning in a Picture or Regular Expression string if you use them. (In some cases, you might need to escape certain special Picture characters with a \ backslash.) In Pictures, you may be able to use () parentheses, or symbols such as ¢ £ or € which are uncommon but are treated as ordinary data characters in these strings. You can also use the single and double “word processing quotes” in the ANSI range of X'91' to X'94' as delimiters. These are just some possibilities for you to explore. The **KEYMAP** facility can be used to map any suitable characters to the desired keys so you can have them readily available to type.

When even such advanced mapping-string techniques are not enough for your needs, an *execute-macro string*, also known as an “**E** string”, may be useful.

Finally, as with any other aspect of SPFLite, if you have questions or need help with this feature, the SPFLite online forum is available to assist you. Of course, if you do that, the first question we will ask is, *Have you read the documentation?* So be sure to start with that important first step.

Examples

For examples of mapping string usage, see [Command Specific Syntax and Examples](#) for more information.

Overview of supported mapping items

Mapping strings support the following types of *mapping items*:

Item Index

[Simple column number](#)
[Zero column number](#)
[Remaining column number – +](#)
[Forward column range](#)
[Backward column range](#)
[End-relative column references – *](#)
[References to result string](#)
[Implied Column References](#)
[Inserted text](#)
[Inserted hex text – 'nn'X](#)
[Inserting literal quotes – SQ / DQ / AQ](#)
[Inserted text ranges – \[x-y\]](#)
[String sets and string pairs – 'A=B' / 'A'='B' / 'A':'B'](#)
[Deleting text – D / DC / RA / RL / RR / M](#)
[Repeating text – RP](#)
[Case modification codes – < > <> ><](#)
[Case alteration commands – UC / LC / SC / TC / IC](#)
[Adjusting the default CASE processing – CC / CT](#)
[Alignment items – L / R / C](#)
[Trimming items – T / TL / TR](#)
[Padding items – P / PL / PR](#)
[Zero-suppression items – Z](#)
[Character-replacement items – RC](#)
[Character-conversion items – AX / XA / EX / XE](#)
[Numeric-conversion items – DD / DX / XD / XX](#)
[Sequence items – SD / SX](#)
[Exchange items – X](#)
[Character-move items – M](#)

Note that for **modifying commands** like *case alteration*, *alignment*, *trimming*, *padding* and *zero-suppression* items, these operations normally work on the **complete** contents of the *result string*. If you wish to perform one of these actions on a **specific** part of the *result string*, that must first be done using a *column-reference* item on that command.

For instance, to copy the first 4 columns of the *source string* and then change **that** to upper case, you would use M"1-4 UC". But if you wanted to copy the entire *source string* to the *result string* (however long it may be), and then make **only** the first 4 columns of the *result string* upper case, you would use M"1+ UC1-4".

To illustrate, if your *source string* contained "abcdwxyz", a mapping string of M"1-4 UC" would give a *result string* of "ABCD", while a mapping string of M"1+ UC1-4" would give a *result string* of "ABCDwxyz". In the first case, you copied only **four** columns (1-4) and upper-cased **all** of them, while in the second case, you copied **all** of the columns (1-8 in this case) but only upper-cased **four** of them.

You can specify a *remaining column-number* item of 1+ to select the entire *source string*. Using 1+ allows the full *source string* to be selected without needing to know its size ahead of time. For example, to perform a zero-suppression on a numeric string that had comma digit-separators, leaving the last 3 positions, you might use a

mapping string of `M"1+ Z,'3"`.

However, most commands operate in either **auto-reference** or in **auto-copy** mode, so that you often will not need to explicitly reference or copy the *source string*. If the *result string* is empty (as it always is at the beginning of your commands in the mapping string), the `Z` command above can be shortened to `M"Z,'3"`. When you do this, the `Z` command detects that the *result string* is initially empty, and first **copies** the *source string* into the *result string* before proceeding.

Simple column number. This is a single decimal number. Leading zeros are allowed and ignored. This selects one character position from the *source string*.

Zero column number. A column number value of zero has a special meaning. When specified as `0`, it refers to all character positions that were not otherwise specified in the mapping string, concatenated together left-to-right as a single value. When specified as `\0`, it refers to all character positions that were not otherwise specified in the mapping string, concatenated together right-to-left in reverse order as a single value. A `\` backslash prefix is not allowed on simple column numbers other than `\0`.

For example, if your data was a date formatted as `12/31/2015`, and you extracted all of the digits using a mapping string of `M"1-2 4-5 7-10"`, that would produce a *result string* value of `"12312015"`, but columns 3 and 6 (where the slashes are) would not have been referenced. If you needed to “gather” all the data you had not referenced up to that point, you would use a column of `0` to get all of them. The mapping string to do that would be `M"1-2 4-5 7-10 0"` and the *result string* value would be `"12312015//"`. The `0` notation is thus a shortcut so you don't have to write all the places you didn't use. It basically means, “everything else”.

In order to determine “all character positions that were not otherwise specified in the mapping string”, all column references in the mapping string **to the left** of where the `0` or the `\0` appears are used, as of that point. If you follow the `0` or the `\0` with additional column references, the `0` or the `\0` will not have a complete list of columns used (because the processing does not look to the right of `0` or `\0` to keep track of referenced columns). This means that, in most cases, if you do this you will not get very useful results. For all but very specialized applications, you will want to make the `0` or the `\0` appear as the **last** column reference in your mapping string, if you use it at all.

Remaining column number +. This is a single decimal number followed by a `+` plus sign. Since `0` is not a valid column number, an item of `0+` is illegal. A remaining column number refers to that column number **and all columns to the right of it, up to and including the last position of the data string value**.

- For example, a remaining column number of `11+` means column `11` and all columns to the right, up to the final character position of the *source string*; if the data value were of length 15, `11+` is the same as `11-15`. When a remaining column number has a `\` backslash prefix, the same columns are referenced, but in reverse order; if the data value were of length 15, `\11+` is the same as `15-11`.
- The `+` plus sign must be followed by a blank, or must be the last item in the mapping string. (You cannot follow the `+` plus sign with another item unless separated with a space, to make sure you have correctly specified your intentions.)
- To select the entire *source string*, a **remaining column number** of `1+` should be used. Because of the **auto-reference** and **auto-copy** features, it will often not be necessary to explicitly copy the *source string* to the *result string* by using `1+` notation.
- When the starting column of the **remaining column number** notation is `1+`, you can omit the `1` if you wish and just specify this as `+` alone. The examples all show `1+` to make clear what their purpose is.

Forward column range. This is two decimal numbers separated by a – dash character, such as **1-4**. There can be no spaces or delimiters between these two numbers other than the dash. The item is in the format *low-high*, and character values are taken from the particular found string in forward order from left to right. Neither column numbers can be **0**.

Backward column range. This is two decimal numbers separated by a – dash character, such as **4-1**. There can be no spaces or delimiters between these two numbers other than the dash. The item is in the format *high-low*, and character values are taken from the particular found string in reverse order from right to left **Note**. Since a backward column range is already in reversed order, it is unnecessary and invalid to combine a \ backslash prefix with a backward column range item. That is, use **20-10** rather than \10-20. Neither column numbers can be **0**.

End-relative column references *. Regular column references assume a character enumeration of 1, 2, 3 ... going from left to right. *End-relative column references* enumerate characters going from right to left as ... 3, 2, 1, by putting an * asterisk after a column number or column range. See the full discussion of [*end-relative column references*](#), and the related topic of [*mixed-mode end-relative column references*](#) for more information.

References to the result string. When a mapping command requires a column-reference operand, you may use **. dot notation**. When this is done, instead of referencing any character locations in the *source data string*, the current, full contents of the *result string* are referenced instead. If the . dot symbol precedes a column reference, like **.1-4**, that column reference is used to select a substring from the current contents of the *result string*. **Note:** If you use *dot notation* before any *result string* characters have been defined, in most cases the *auto-copy* and *auto-reference* features will populate the *result string* with a copy of the *source string*.

Implied column references. When you use a *copying command* like AX, it requires a reference to the *source string*, or to the *result string* via a *dot notation*. When you use a *modifying command* like UC it will modify the entire *result string*, unless you provide a column range for portion of the *result string* that you wish to upper case. For such commands, **if you omit a column reference**, the following **assumptions** will be made:

- For a **copying command** like AX, it will assume you want to reference and copy the entire *source string*. So, AX by itself means the same as AX1+. This feature is called **auto-reference**.
- For a **modifying command** like UC, it requires some value to be present in the *result string*, which it then operates on. If a *modifying command* is issued before anything has been copied into the *result string*, **it is assumed** that you wanted to apply that command to the complete value of the *source string* (since you cannot modify the *source string* directly). Therefore, when the *result string* is empty, the **modifying command** will cause the value of the *source string* to be **automatically copied** into the *result string*, since that is most like what you would need to do, and because the command cannot operate without some data in the *result string*. (Modifying commands have to have *something* to modify.) This feature is called **auto-copy**. If you want some different value in the *result string*, you will have to explicitly copy it before issuing any *modifying commands*. The *auto-copy* feature will not be done if any data has been placed into the *result string*.

The [Quick Reference](#) shows you for each command whether it is a **copying command** or a **modifying command**.

Inserted text. An inserted text item consists of quoted text, enclosed by any valid SPFLite quote, such as "abcd". Inserted text strings are appended to the end of the *result string*.

- Due to standard SPFLite syntax rules, you must use a quote type **other than** the one being used to enclose the **M** string as a whole. For example, if you used double quotes (") to enclose the **M** string,

inserted text items must be enclosed in single quotes (') or accent quotes (`) but they cannot use double quotes ("). To define an inserted text quote-character that is the same as its enclosing quotes, that quote must be **doubled**, or else you must use the 'other quote type not otherwise being used (in a different string). Quoted string operands in mapping commands do not use an "escape" mechanism like backslashes. See [Inserting literal quotes](#) below for more information.

- When a literal like `` appears in a mapping string with no other text before or after it, it represents a zero-length string. Depending on the command where it is used, it may be treated the same as a single blank, it may be ignored, or it may be an invalid string operand.
- As you would expect, it is an error to begin an inserted text string with a quote character without finishing it by the time the end of the **M** string is reached. Inside an inserted text string, values are literal characters stored in the *result string* value as-is. When a quoted string is part of a command code, see the discussion of the particular command for more information on how it handles its string operands.
- Case *modification codes* (discussed below) do not apply to inserted text literals; the literal values are always inserted into the data exactly as specified. Due to the same standard SPFLite syntax rules, you cannot directly insert text that contains the quotes that enclose the **M** string. You could accomplish this by [Inserting literal quotes](#), or with [Inserted hex text](#), described next.
- Inserted text can have a replication factor as a suffix on the string. A literal of ' / ' 3 will produce the same value as ' / / / '. This feature only works for "stand-alone" text to be inserted into the *result string*; you cannot use a replication factor on a string that is an operand of a command code.

Inserted hex text 'nn'X. This serves the same purpose as regular *inserted text*, but you specify the value as two or more hex digits **0-9**, **A-F** and **a-f**. Any underscores inside of *inserted hex text* are ignored; the remaining characters must be an even number of valid hex digits. In a mapping string, an *inserted hex text* literal must be identified only with a trailing **x** or **x** code, like '77'x, not like x'77'. If you needed to insert a literal quote as data which was the same as the outer quotes used on the **M** string, you could use a hex text value. For instance, if your **M** string were enclosed in (") double quotes, you could insert a double-quote literal using '22'X, the ASCII value of a double-quote.

Inserted hex text can have a replication factor as a suffix on the string. A literal of ' 41 'x3 will produce the same value as '414141'x, which is 'AAA'. This feature only works for "stand-alone" text to be inserted into the *result string*; you cannot use a replication factor on a string that is an operand of a command code.

Inserting literal quotes SQ / DQ / AQ. Because it can be awkward to insert the quote characters that are used to enclose the outer M string, there are three command codes that will do this, which are SQ, DQ and AQ. These will insert a Single Quote, Double Quote or Accent Quote, respectively. You can place a replication factor on these commands; AQ2 will produce `` two accent quotes.

Note: Literal-quote command codes are **neither** "copying commands" **nor** "modifying commands" in the usual sense. They simply append one or more quotes at the end of the *result string*. This means that the *auto copy* and *auto reference* features **do not apply**. If you wanted to enclose the *source string* in double quotes, you could do this with a mapping string of **M"DQ 1+ DQ"**.

Inserted text ranges – [x-y]. This is specialized form of *inserted text* in which one or more pairs of Ansi characters are generated, and appended to the end of the *result string*. An inserted text range has the form [x-y] where **x** and **y** are any single Ansi characters. Example: [0-9] would produce the same as '0123456789'. If the range is in reverse collating order, the characters produced are reversed; [9-0] would produce the same as '9876543210'. You can specify multiple ranges by a comma-separated list within the brackets. Example: [A-Z,a-z,0-9]. As long as the correct format is observed, with commas, dashes and brackets in their proper places, the characters of **x** and **y** can be **anything**, including commas, dashes and brackets **themselves**. (This will look

strange, but it will work.) If you need to insert a single character within a larger range, create a range item that uses the same character twice. For example, to create a list of digits in ascending order, then a slash, then the digits in descending order, specify this as [0-9,/-,9-0]; that will produce '0123456789/9876543210'. *Inserted text ranges* do not support replication factors. (If you needed that, the Repeat command code RP might be useful.)

String sets and string pairs 'A=B' / 'A='B' / 'A':'B'. Certain commands such as RC require a string operand with two values – a *first* value and a *second* value, separated by an = equal sign. When this is specified as a **single** string, it is called a **string set**. For example, the command RC'ABC=def' contains a *string set*. When one or both of the values themselves contain = equal sign characters as data, you must specify these as **two** quoted values using a **string pair**. Suppose you wanted the RC values to be **A=B** and **X*Y**. To do that, you specify it with a *string pair* of 'RC'A=B='X*Y'.

- The only difference between a **string set** and a **string pair** is whether there is **one** quoted value or **two**. You only need a “pair” if the *first* or *second* values contain embedded equal signs as data. It helps with the documentation for us to be able to give each specify format a “name”. Otherwise, these values are used internally by the commands in exactly the same way, since a command is only concerned about the *values* of *first* and *second*, not on the syntax of **how** they got defined.
- Either or both quoted values of a *string pair* may optionally contain a string type code of C, T or X. Consult the description for each mapping command that allows *string pairs* as to how they handle these string type codes. Because there are a large number of combinations of type codes C, T or X (or none) that could be on both sides of the *string pair*, consult the Quick Reference for how these combinations are treated. It contains a comprehensive table of all combinations of these codes, and how they are processed.
- Both values of a *string pair* must be quoted, but the left and right sides do not have to use the same **kind** of quotes.
- When the delimiter between a *string pair* is a : **colon** instead of an = equal sign, the command is instructed to treat the string values as both **case sensitive** and **case conformant**. This is a shortcut for putting string type T on both sides.

Deleting text D / DC / RA/ RL / RR / M. You will usually not need a special command to delete text. To accomplish a deletion of characters from the *source string*, you can simply omit any reference to those columns of source data from the mapping string. However, if do you have some need to delete characters from the *result string*, this can be done with a Delete mapping item code of **D** followed by a required *column-range*. When you use the **D** command code, the *column-range* always refers to character positions within the *result string*. You can delete “classes” of characters, such as “all commas, periods and parentheses” by using the Delete Characters command **DC**. You can replace or delete a specified string wherever it is located, with the String Replacement commands **RA**, **RL** and **RR**. You can delete text in the process of moving it from one part of the *result string* to another, using the Move command **M**.

Repeating text RP. The command code RP will repeat the contents of the *result string* any number of times. RP by itself will duplicate the *result string*. Example: A mapping string of M"Abc` RP" will cause the *result string* to contain AbcAbc. RP will take a duplication factor. Example: A mapping string of M"Abc` RP2" will cause the *result string* to contain AbcAbcAbc. The duplication factor indicates how many *additional* copies of the *result string* will be made. If RP appears by itself as the first thing in the mapping string, the *result string* will get two copies of the *source string* (due to the *auto-copy* feature being applied); first, the *source string* would get automatically copied to the *result string*, and then RP will repeat the *result string* so there are now two copies of it. Note that the numeric operand for RP is a duplication factor, not a *column reference*.

Case modification codes < > <> ><. Normally, the character casing of the *result string*, after mapping is completed, is the same as the *source string*. If desired, the casing can be altered by a *case modification code*. These codes are detected as the mapping string is scanned from left to right. The codes define a *case*

modification state. Once a *case modification state* is defined, that state remains in effect to the end of the mapping string or until altered by another *case modification code*. *Case modification codes* do **not** apply to inserted text literals; these literal values are always inserted into the data as-is exactly as specified. A *case modification code* must be followed by a blank. Because a *case modification code* affects commands that follow it, placing a *case modification code* at the **end** of a mapping string is legal but will have no effect. Mapping strings support the following *case modification codes*:

- *Upper case code* > causes subsequent mapped characters to be converted to upper case.
- *Lower case code* < causes subsequent mapped characters to be converted to lower case.
- *Invert case code* >< causes the case of subsequent mapped characters to be inverted, so that upper case becomes lower case and vice-versa.
- *As-is case code* <> causes subsequent mapped characters to be copied as-is without case conversion. By default, all mapping strings are treated as if they began with an initial as-is code of <> so that the case of mapped characters is not altered.
- If a *case modification code* of > (convert to upper case) or < (convert to lower case) is in effect prior to the **AX** or **EX** code, those *case modification states* will be respected, and hex-digit characters will be rendered accordingly. If a *case modification code* of > or < is **not** in effect prior to the **AX** or **EX** code, hex-digit characters will be rendered in upper case.

Consult the [Quick Reference](#) to determine how the various command code process an active *case modification state*.

Note that a **modifying command** will **auto-copy** the *source string* to the *result string* if, at the time the command is encountered, there is **as yet** nothing in the *result string*. If there is an active *case modification state* (other than the “as-is” state that doesn’t change anything), and such an **auto-copy** occurs, that copied data will have its case **modified** as you requested.

For example, if your *source string* contains “◦◦ abc ◦◦” (the “◦” symbol represents one blank) and you issue a mapping command of **M"> L**”, at the point the L command is reached, there is as yet nothing **in** the *result string*. Because the L command is a **modifying command** that will **auto-copy**, the *source string* is first copied to the *result string* before L acts on it. During that copying process, it is noted that the *case modification state* of > is in effect, so that when the data is copied, it gets changed to upper case before L acts on it. Thus, L will see the *result string* as containing “◦◦ ABC ◦◦”, which it then left-justifies as “ABC ◦◦◦◦”.

Case alteration commands UC / LC / SC / TC / IC. Five case alteration commands are defined: **UC**, **LC**, **SC**, **TC** and **IC**. These will alter the current contents of the *result string*. Just like their Primary Command and Line Command counterparts, the first four mapping commands alter the character case of the *result string*, comparable to how those primary and line commands do it (Upper Case, Lower Case, Sentence Case and Title Case). The code **IC** is used to Invert Case, the way the *case conversion code* >< operates.

Why have these commands, when the *case modification codes* described above are available? Case modification codes **do** allow for precise control over the casing of data selected by specific column ranges. However, you may need a case alteration command for data extracted from locations you do not know of in advance. In addition, the command codes **SC**, **TC** and **IC** perform operations that would difficult to accomplish with case modification codes. **SC** capitalizes the first “word” of each “sentence”, while **TC** capitalizes the first “letter” of each “word”.

These operations depend on the following definitions: A “letter” is a character whose upper case and lower case values are different. A “word” is a span of “letters” that begins with the beginning of the string or a blank, and ends with the end of the string or a non-letter character. A “sentence” is one or more “words”, where the first “word” of a “sentence” is the first “word” of the string, or any “word” preceded by both a period and a space, in that order. These rules implement an algorithm that works well for properly formatted English text, but if your data is out of the ordinary, **TC** and **SC** may or may not suit you. You should test these codes on some sample data to make sure they meet your needs.

Adjusting the default CASE processing CC / CT. Certain mapping items that involve comparisons to character data in the *source string* are sensitive to the PROFILE CASE setting. These mapping items are the *character replacement* command **RC**, the *string replacement* commands **RA**, **RL** and **RR**, and the exchange-string command **X**. By default, whenever a character comparison takes place, the current **PROFILE CASE** setting determines how it is done. If **PROFILE CASE C** is in effect, comparisons are done as case-sensitive, and if **PROFILE CASE T** is in effect, comparisons are done as case-insensitive. There are two ways to override this.

- First, commands that take strings which are being compared can have a **C**, **T** or **X** suffix. If a string has a suffix of **C** or **X**, strings are compared as case-sensitive, and if a string has a suffix of **T**, strings are compared as case-insensitive.
- Second, if a command string does **not** have a **C**, **T** or **X** suffix, you can use the *case-override items* **CC** and **CT** to override the **PROFILE CASE** default. If the command **CC** appears in the mapping string, comparisons for any command that follows it are done in case-sensitive mode, as if your PROFILE was set to **CASE C**. If the command **CT** appears in the mapping string, comparisons for any command that follows it are done in case-insensitive mode, as if your PROFILE was set to **CASE T**. The **CC** and **CT** commands are handled left-to-right as they appear in a mapping string, and remain in effect for that string unless followed by another **CC** or **CT** command. You can use the **CC** or **CT** command to set the case-handling just once, rather than having to use string type codes everywhere commands appear in your mapping string. The **CC** and **CT** commands do not take any string or number operands.
-

Alignment items L / R / C. An alignment item consists of the letter **L**, **R** or **C** followed by an *alignment specification*. An alignment item is used to left-justify, right-justify, or center the *result string*.

Trimming items T / TL / TR. A trimming item consists of the codes **T**, **TL** or **TR** followed by an optional *trimming specification*. A trimming item is used trim off one or more blanks (or other specified character) from one or both side of the *result string*.

Padding items P / PL / PR. A padding item consists of the codes **P**, **PL** or **PR** followed by a *padding specification*. A padding item is used append one or more blanks (or other specified character) on to one or both side of the *result string*.

Zero-suppression items Z. A zero-suppression item consists of the letter **Z** followed by an optional *zero-suppression specification*. A zero-suppression item is used to suppress leading zeros from a numeric value in the *result string*, and replace them with spaces.

Character-replacement items RC. A character-replacement item consists of the code **RC** followed by a *character-replacement specification*. A character-replacement item is used to replace one set of characters, using a character translation table.

Character-conversion items AX / XA/ EX / XE. A character-conversion item consists of the codes **AX**, **XA**, **EX** or **XE**, followed by a *column-range specification*. A character-conversion item is used to convert between character display-data and their hex-digit equivalents. For example, the hex equivalent of Ansi '1' is '31'.

Numeric-conversion items DD / DX / XD / XX. A numeric-conversion item consists of the codes **DD**, **DX**, **XD** or **XX**, followed by *numeric formatting information*. A numeric-conversion item can alter the format of a numeric value by adding or removing leading zeros and by changing radix between decimal and hex. It is possible to calculate derived values from the initial numeric value, such as constant multiples or offsets from the original number, using a *calculation operand*.

Sequence items SD / SX. A sequence item consists of the codes **SD** or **SX**, followed by a *sequence specification*. A sequence item is used to generate a sequence of numbers in decimal or hex in various formats. When a CHANGE ALL command uses an M string, the first string that is found is assigned an enumeration of 1; the second has an enumeration of 2, etc. It is possible to calculate derived values from the enumeration, such as constant multiples or offsets from the original sequence number, using a *calculation operand*.

Exchange items X. An exchange item consists of the code **X** followed by an *exchange specification*. An exchange item is used swap two different strings at the same time, such as replacing all occurrences of 'one' with 'two' and vice-versa.

Character-move items M. A character-move item consists of the code **M** followed by a *character-move specification*. A character-move item is used to move substrings from one point to another within the *result string*.

Mapping String Command Topics

Topic Index

[Specifying an end-relative column reference item](#)
[Specifying a mixed-mode end-relative column reference item](#)
[Specifying an external mapping string item – **SV**](#)
[Specifying a dynamic mapping string item – **DV**](#)
[Specifying an alignment item – **L / R / C**](#)
[Specifying a trimming item – **T / TL / TR**](#)
[Specifying a padding item – **P / PL / PR**](#)
[Specifying a character-reversal item – **RV**](#)
[Specifying a zero-suppression item – **Z**](#)
[Specifying a character-replacement item – **RC**](#)
[Specifying a string-replacement item – **RA / RL / RR**](#)
[Specifying a character conversion item – **AX / XA / EX / XE**](#)
[Specifying a numeric conversion item – **DD / DX / XD / XX**](#)
[Specifying a sequence item – **SD / SX**](#)
[Specifying an exchange item – **X**](#)
[Specifying a character-move item – **M**](#)
[Specifying a delete character-columns item – **D**](#)
[Specifying a delete character-values item – **DC**](#)
[Specifying a calculation operand](#)

End-relative column references

When the Picture or Regular Expression *find string* of your **CHANGE** command locates data of varying lengths, it may be more convenient to refer to data positions **relative to the end of the string**, rather than **relative to the beginning**. Suppose we find a string containing **abcPDQxyz**. (Remember that the string that is found is the *source string*.)

The **regular enumeration** of the *source string* value is as follows:

```
abcPDQxyz
123456789
```

End-relative enumeration of this same *source string* value is the reverse of that:

```
abcPDQxyz
987654321
```

Now, if every string you found were exactly 9 positions long, and you wanted the last 3 positions, you could easily get them with a column range of **7-9**. But, if the strings varied in size from **5** to **20** characters, what could you use to get those last 3 positions? Without using end-relative notation, there is no easy way to do it.

To specify an **end-relative column reference**, you use a single column number, or a column range with two number separated by a **-** minus sign, and then follow it with an ***** asterisk.

Examples, using the *source string* noted above:

- To reference the rightmost character of the example data ("z") use **1***
- To reference the third character from the right ("x") use **3***
- To reference the rightmost 3 characters ("xyz") use **3-1***
- To reference the rightmost 3 characters in reverse order ("zyx") use **1-3***

Note carefully: When an **end-relative column range** is used, a notation of **first-second*** selects columns in **reverse** order when **first** is **smaller** than **second** (like **1-3***), and in **forward** order when **first** is **larger** than **second** (like **3-1***). Because of this, you must use this notation carefully and not get these confused.

The column ordering is the **exact opposite** of how **normal column-range** references are done, where a range like **1-3** is forward order, and **3-1** is reverse order. This is necessary in order to be consistent with how the columns are numbered in reverse order (as shown above), when **end-relative** column numbering is used.

Example: To select the last 3 positions of a variable length field and convert them to upper case, use this:

```
CHANGE R' [A-Za-z]+ M'> 3-1* WORD ALL
```

Example: To select the last 3 positions of a variable length field and convert them to upper case, **and reverse their order**, use this:

```
CHANGE R' [A-Za-z]+ M'> 1-3* WORD ALL
```

End-relative notation can be used to on any command that accepts a *column reference*.

End-relative notation cannot be combined with **+ remaining-column** notation, and **0** cannot be used as a column number. (Those capabilities are not really needed with **end-relative** notation, since you can already

reverse the columns selected by switching the two numbers in a range.)

The range of column numbers that can be used for **end-relative notation** is the same as for **regular notation**:

- If an **end-relative** single column number is higher than the length of the *source string*, no data is selected.
- If an **end-relative** single column range has both numbers higher than the length of the *source string*, no data is selected.
- If an **end-relative** single column range has one column number higher than the length of the *source string* but not both, the higher number is effectively reduced to be equal to the length of the *source string*.

The [Mapping Strings Quick Reference](#) has several examples of how all the various column reference types are specified and what they mean.

Mixed-mode end-relative column reference item

The standard and **end-relative** notations discussed above work well for most cases, but a problem arises when you want to reference **both** ends of a variable-sized item. For example, suppose you had some string that varied in size, and you wanted to reference **all but** the first two and last two positions. (That is, you wanted to start with the third character in, from **both** ends.) You can **omit** the first 2 positions by using 3+ to skip the first two, and you can **get** the last two by 2-1* but there is no way to get a string that omits **both** sets of characters on both ends.

This is a situation you will not encounter often, but if you do, **standard column notation** will not handle this.

Instead, you must use **mixed-mode end-relative notation**. In this notation, **the first column number you specify is a 'normal' one, relative to the left side, and numbered from left to right from 1 up. The second number is end-relative, numbered right to left starting with 1 and up leftward, as is done for standard end-relative columns**. Because the second number is **end-relative**, it must be followed by an * asterisk, to make clear your intentions. By using **mixed-mode end-relative notation**, you can define a column range that is relative to both ends at the same time.

Mixed-mode column ranges use a separator of / **slash** rather than – **dash** between the two numbers. In our example, reference **all but** the first two and last two positions of some string, you would use 3/3*. In a string of length 10, this would select the same columns as 3-8 would.

When you need to use **mixed-mode** columns and also want to reverse the order of the columns, use a separator of \ **backslash** rather than / **slash** between the two numbers, such as 3\3*. In a string of length 10, this would select the same set of reversed columns as 8-3 would. That is, first columns **3** to **8** would be selected, and then the character order would be reversed.

The [Mapping Strings Quick Reference](#) has several examples of how all the various column reference types are specified and what they mean.

External mapping string item – **SV**

Because mapping strings can sometimes be long and intricate to specify, you may benefit from saving a copy of them in a text file, to have them ‘on hand and available if you need to use them again. However, you would still need to cut-and-paste such a string from your text file into the mapping command that you issued, which may not be the most convenient thing to do.

You could also define entire mapping strings as **SET** variables. When you did that, you would put everything, including the M type code, as the value of the variable, and then reference that by name on the command line with a leading = equal sign. When preparing such **SET** variables, you will find it easier to issue a plain **SET** command, and then type in your values in the Set Edit screen, rather than typing it all on the primary command line.

The external mapping string facility described here would be used when you only want “pieces” of your mapping string replaced by outside definitions rather than the entire mapping string. By using an external mapping string, you have the benefit of maintaining a copy of your frequently used mapping strings without having to do a cut-and-paste of them. When you do this, you can store your external strings as SPFLite **SET** variables.

Syntax of an *external mapping string item*:

SV *symbol-name*

SV:

Introduces the external mapping string item. **SV** directs the mapping-string processor to use an SPFLite **SET** variable as the source for further mapping command items.

Symbol-name:

A quoted-string operand that defines the name of the variable to be referenced. It is an error if the variable in question is not defined. **It is not an error if the variable in question exists but has a text value of zero length or only contains blanks.** That is a technique you could use to make certain commands like “no operation” commands, especially if you have ‘nested’ these commands. It would be a technique to selectively enable or disable certain commands. See comments below about nesting.

If for some reason defining value that is empty on the right-hand side of the = equal sign is a problem, you can use a lone ; semicolon, which is treated as equivalent to a blank when unquoted, and is ignored.

The *symbol name* cannot be missing or blank or contain any characters other than letters, digits or underscores.

When you specify a *symbol name*, a prefix is added to the name you supply. If you issue a mapping command of SV'ABC', the SPFLite **SET** variable that is searched for is MAPSTR.SV.ABC.

Once the mapping commands found in the mapping variable are processed, the original mapping commands after your **SV** command resume where they left off. In that sense, the SV mapping commands are like #INCLUDE statements in a programming language.

It is permitted for **SV** commands to nest, up to 10 levels deep. It is an error to refer to these variables in a way that is “recursive”, whether directly or indirectly. The mapping string logic will detect a recursive reference and halt the processing with an error message if this occurs. Recursion is defined as any SET variable being referenced again before its content has been fully “consumed”, or if for any reason the nesting level exceeds 10.

When you use the SPFLite **SET** facility, simply quote the value exactly as you want to store it. For example, if you want to make a **SET** name for ABC, and define it as the mapping command **RA 'ABC=DEF'**, issue this at the primary command line:

```
SET MAPSTR.SV.ABC = "RA'ABC=DEF"
```

When you use a **SET** variable defined this way, your mapping string would reference it with an item of SV'ABC'.

Dynamic mapping string item – DV

This is an advanced technique, somewhat related to the SV commands. Because you could run into problems if this command is used incorrectly, you should be sure you fully understand the mapping string facility before attempting to use this command. It is highly recommended that you test this technique first, using the [Mapping Test Program](#), before attempting to use it with your production data.

The purpose of the DV command is to create a dynamic mapping command.

Syntax of a *dynamic mapping string item*:

DV [*delete-option*] *column-reference*

DV:

Introduces the dynamic mapping string item. **DV** stands for Dynamic Value, as compared to Set Value (**SV**).

delete-option

An optional quoted string operand. If present, it contains a single letter specifying what is done with the *result string* prior to the DV command finishing. The letter is case-insensitive.

- 'K' means to keep the *result string* as-is without change.
- 'D' means to delete the portion of the *result string* that was referenced by the *column-reference* operand. This is the default if the *delete-option* is omitted.
- 'E' means to erase the entire *result string* so that it contains a null (empty) string as it does when mapping string is first processed.

Column-reference:

A standard column reference. Only the *result string* is referenced.

A dynamic command value is formed by creating, in the *result string*, any suitable mappings commands as text values. You do this using any desired mapping commands to help you build up one or more **new** mapping commands **as data**, by whatever means is needed. Such commands to build up and create these strings appear in your main mapping string before the DV is specified.

You then use the *column-reference* operand of the DV command to select one or more columns from the result string as your “dynamic value”, taking care that the text you built up and selected is in fact valid as one or more mapping commands. This text is “inserted” into the original mapping string at the point where the DV command appeared.

Because any problems with your inserted commands, such as unmatched quotes or other syntax errors, could cause the mapping string as a whole to malfunction, you should exercise great care using DV commands.

Prior to executing the commands you inserted with the DV command, some or all of the current contents of the *result string* can be deleted, depending on the *delete-option*

operand you specify. As noted, if you omit the *delete-option* operand, the default is to remove the portion of the *result string* text that was submitted for execution.

Once the mapping commands you have provided are processed, the original mapping commands after your **DV** command resume where they left off. In that sense, the **DV** mapping command is like an #INCLUDE statement in a programming language.

If the text you select from the *result string* is blank or empty, it is not an error. The blank text that is submitted by **DV** for execution will simply do nothing.

Unlike the **SV** command, you cannot “nest” a **DV** command. That is, the dynamic value you submit to the mapping system to be executed cannot itself have a **DV** command. If this detected, execution of the mapping string as a whole is halted and an error message produced.

Why would you want to use a DV command?

The most likely reason is that you need to perform some action that is dependent on the data you have found. You may have data files that contain mapping string command codes, and you want to directly execute them. You might also wish to construct an elaborate *calculation operand* that sets one or more *calculation variables*. There are doubtless many uses for this facility, limited only by your imagination.

Specifying an alignment item – L / R / C

An *alignment item* is used to left-justify, right-justify, or center, the *result string*. Recall from the discussion above that the *result string* is built up, a piece at a time, in a left-to-right manner. When an *alignment item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to align the 'final *result string*' in a certain way, you would place the *alignment item* last (right-most) in the mapping string.

Because the action of an *alignment item* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of an *alignment item*, since the data characters in the *source string* remain undisturbed in their original locations.

An alignment item consists of the letter **L**, **R** or **C** followed by an *alignment specification*.

Alignment is always performed with respect to some specified field length. That is, when you left justify a string, it is done within some given area, say, a length of **9**. If your original data is longer than **9**, the excess is trimmed off, while if it is shorter, the data is padded on the other side.

Suppose you have a *result string* with some blanks on each side of it, and you wanted this string to be left justified in its current size. You do this by omitting the length, or by specifying a length of **0**. When this is done, the *result string* is effectively first **trimmed** of leading and trailing blanks (or other character you specify), and then left-justified within a field that is the size that the *result string* originally was before it got trimmed.

For example, if the *result string* contains "xy" and you use a command of **L** or **L0**, (noting that the *result string* is initially of length **4**) the string is first trimmed to 'xy', and then left-justified within a length of 4, to produce 'xyoo', the same as if the command had been specified as **L4**. (The "o" symbol represents one blank.)

When alignment causes padding to occur, the default pad character is a blank. You can optionally specify a different padding character as a literal value.

No spaces or characters not part of the syntax can appear within a single *alignment item*, unless appearing inside of quotes as a literal *pad-character* value.

In the discussion below, the command **L** for left-justifying is used as an **example** to make the explanation easier to follow; the same rules apply to **R** and **C**. A literal length of **3** or **9** is used for similar reasons, to make the examples easier to read. An example of an optional padding character of 'x' is shown, where 'x' can be any single character.

Syntax of an *alignment item*:

{ **L** | **R** | **C** } [*pad-character*] *field-size*

L | **R** | **C**:

L, **R** or **C** refer to Left-justifying, Right-justifying, or Centering, respectively.

Pad-character:

Any single character in quotes, as described above under *Inserted text*. The pad character is optional, and if omitted, blanks are used for padding. If the pad string is longer than one character, only the left-most is used. If the pad string is specified as two consecutive quotes, padding is done with a single blank, just as if you omitted the pad character altogether.

Field size:

This is similar to a column-reference specification, with a few differences.

L9

L'x'9

The current *result string* is Left-justified, with a fixed length of **9**. If the *result-string* was longer than **9**, excess characters are truncated on the right. If the *result-string* was shorter than **9**, extra padding characters are added on the right to force the length to be **9** characters. If *pad-character* 'x' was specified, that is the character used for padding; otherwise padding is done with blanks. If the command is specified as **L** or **L0**, the *result string* is justified within the bounds of its current size.

L3-9

L'x'3-9

The current *result string* is Left-justified, with a minimum length of **3** and a maximum length of **9**. If the *result-string* was longer than **9**, excess characters are truncated on the right. If the *result-string* was shorter than **3**, extra padding characters are added on the right to force the length to be at least **3** characters. If *pad-character* 'x' was specified, that is the character used for padding; otherwise padding is done with blanks. It is legal to specify a minimum length of **0**. It is illegal to specify a minimum length greater than the maximum length. It is illegal to specify **Ln-0** (where **n** is any number) since that would discard the entire *result string*.

L-9

L'x'-9

The current *result string* is Left-justified, with a maximum length of **9**, but no minimum length. If the *result-string* was longer than **9**, excess characters are truncated on the right. If the *result-string* was shorter than **9**, no padding takes place, and the value is allowed to be remain shorter than **9** characters. It is illegal to specify **L-0** since that would discard the entire *result string*.

Note 1. Because padding does not take place with alignment items in this form, any padding string that you may specify is allowed but will be ignored. Thus, **L'x'-9** is processed exactly the same as **L-9**, which is why this is shown above as grayed-out.

Note 2. An *alignment item* of **L-9** operates the same way as **L0-9**.

L9+

L'x'9+

The current *result string* is Left-justified, with a minimum length of **9**, but no maximum length. If the *result-string* was longer than **9**, no padding takes place, and the value is allowed to be remain unchanged in its current length. If the *result-string* was shorter than **9**, extra padding characters are added on the right to force the length to be at least **9** characters. If *pad-character* 'x' was specified, that is the character used for padding; otherwise padding is done with blanks. It serves no purpose to specify a minimum length here of **0**, using **L0+**, since every string is at least zero characters long; the string would not be changed.

When the *alignment code* is **L**, data is aligned left, and padding and truncation take place on the right.

When the *alignment code* is **R**, data is aligned right, and padding and truncation take place on the left.

When the *alignment code* is **C**, data is centered within the specified field. This is handled somewhat differently than left- or right-aligned data. If centering of data within a field results in an odd number of 'extra positions', centering is done in such a way that the number of remaining extra positions is divided by two, and the 'short half goes on the left side, while the 'long half goes on the right side. So, if you have a 4-character field and you are centering it in a 9-position field, there are 5 extra positions to deal with. Since 5 is odd, we divide it into 2 and 3.

The result string will have 2 positions of padding on the left, then 4 positions of data, then 3 positions of padding on the right.

Trimming items – T / TL / TR

A *result-string* is *trimmed* by removing leading or trailing pad characters (or both). Recall from the discussion above that the *result string* is built up, a piece at a time, in a left-to-right manner. When a *trimming item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to trim the ‘final’ result a certain way, you would place the *trimming item* last (right-most) in the mapping string.

Because the action of a *trimming item* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of a *trimming item*, since the data characters in the *source string* remain undisturbed in their original locations.

A trimming item consists of the codes **T**, **TL** or **TR** followed by a optional *trimming specification*.

When trimming causes the *result string* to be truncated, the default pad character that is searched for is a blank. You can optionally specify a different padding character as a literal value. Be mindful that for the Trim commands, “padding” defines, not the padding to be **added**, but the *existing padding* that is going to be **removed**.

If trimming results in all characters being removed, the result is an empty (zero-length) string. Thus, if you request all blanks to be removed, and the value contains only blanks, the result will be a zero-length string. If that is not the result you want, you could ‘correct’ that by following the trim request with an alignment item that specifies the desired field length, such as a Pad command, or you could use a text-insertion command to add back some value if desired.

No spaces or characters not part of the syntax can appear within a single *trimming item*, unless appearing inside of quotes as a literal *pad-character* value.

Syntax of an *alignment item*:

{ **T|TL|TR** } [*pad-character*] [*maximum-trim*]

T|TL|TR:

Refers to the type of trimming that takes place:

- **T** trims *pad characters* from both the left and right sides
- **TL** trims *pad characters* from the left side
- **TR** trims *pad characters* from the right side

Pad-character:

Defines the character to be searched for and removed. May be any single character in quotes. If omitted, a *pad-character* of blank is used. See *Alignment Item* above for more information on the syntax of this part.

Maximum-trim:

A decimal number that defines the maximum number of padding characters that can be removed from either side of the *result string*; if used with the **T** command code, the same maximum is applied individually to both sides. If omitted, all possible padding characters that are present are removed. It is illegal to specify a *maximum trim* length of **0**, since that would request the trimming operation to do nothing.

Padding items – P / PL / PR

A *result-string* is *padded* by adding leading or trailing pad characters (or both). Recall from the discussion above that the *result string* is built up, a piece at a time, in a left-to-right manner. When a *padding item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to pad the ‘final *result string* in a certain way, you would place the *padding item* last (right-most) in the mapping string.

Because the action of a *padding item* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of a *padding item*, since the data characters in the *source string* remain undisturbed in their original locations.

A padding item consists of the codes **P**, **PL** or **PR** followed by a optional *padding specification*.

By default, the pad character used for padding is a blank. You can optionally specify a different padding character as a literal value.

No spaces or characters not part of the syntax can appear within a single *padding item*, unless appearing inside of quotes as a literal *pad-character* value.

Syntax of an *alignment item*:

`{ P|PL|PR } [pad-character] [pad-amount]`

P|PL|PR:

Refers to the type of padding that takes place:

- **P** adds *pad characters* on both the left and right sides
- **PL** adds *pad characters* on the left side
- **PR** adds *pad characters* on the right side

Pad-character:

Defines the pad character to be used. May be any single character in quotes. If omitted, a *pad-character* of blank is used. See *Alignment Item* above for more information on the syntax of this part.

Pad-amount:

A decimal number that defines the number of padding characters that are to be added to either side of the *result string*; if used with the **P** code, the same number of pad characters are added to both sides. If omitted, the *pad-amount* length is treated as **1**. It is illegal to specify a *pad-amount* length of **0**, since that would request the Pad command to do nothing.

Character reversal item – RV

A *character reversal item* reverses the order of characters in the *result string*.

Syntax of a *character-reversal item*:

RV [*column-reference*]

RV:

Introduces the character-reversal item.

Column-reference:

Defines the columns within the *result string* to be reversed. This can be any standard *column reference* operand, including *end relative notation* with an * suffix. You cannot specify a column of **0** or **\0**.

If *column-reference* is omitted, the order of entire contents of the *result string* are reversed.

Since column-reference mapping items and most command codes already allow for string reversal, this command would only be needed in special circumstances, such as when the *result string* had been built up with a complex set of mapping items, and then you needed to reverse the final value of it. In most cases, you will likely not need to use **RV** for this. It is possible for the *column reference* to also involve column reversal, but doing so would serve no purpose.

Zero suppression item – Z

A *result-string* is *zero-suppressed* by removing leading **0** characters from the value and replacing them with spaces. Recall from the discussion above that the *result string* is built up, a piece at a time, in a left-to-right manner. When a *zero-suppression item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to zero-suppress the ‘final result a certain way, you would place the *zero-suppression item* last (right-most) in the mapping string.

Because the action of a *zero-suppression* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of a *zero-suppression*, since the data characters in the *source string* remain undisturbed in their original locations.

There is no provision for zero suppression to act other than to replace suppressed characters in the *result string* by spaces. Because of this, the *result string* will always have the same length after zero suppression is done as it had beforehand. If you need some other result, such as replacing leading zeros with a special character like * or to remove leading zeros altogether without replacing them with spaces, some other mapping codes might be used to achieve these effects.

If you need something more elaborate, such as floating +/- signs, floating dollar signs, enclosing negative numbers in parentheses or suffixing them with DB/CR codes (the sorts of things you can do in a COBOL program with a PICTURE clause), an *execute-macro string*, also known as an “**E** string”, may be useful. It may also be possible with a careful combination of trimming, padding, text insertion, and Move and Delete commands, to achieve some of these results.

Zero suppression can be applied to any character string, but it will make the most sense to use this on strings that are already formatted as numeric values. For example, suppose you had the string “0,012.34”. If this were zero-suppressed, you would probably want any “insignificant” commas removed, but you would not want the decimal point and digits after it removed, even if they were zero. That is, you would most likely want the string changed to “**•••12.34**” (where “**•**” represents blank positions).

A zero-suppression item consists of the letter **Z** followed by a optional *zero-suppression specification*.

Syntax of a *zero-suppression item*:

Z [*ignore-characters*] [*minimum-length*]

Z:

Introduces the zero-suppression item

Ignore-characters :

Defines a list of one or more leading non-significant characters to be ignored while performing zero suppression. Once a “significant” character (any character in the range of **1-9**, or any character not in the ignore-list) is encountered while scanning the data left-to-right, that digit and all data to the right of it is considered as “significant”. By default, leading **0** and blank characters are ignored; *this assumption cannot be changed*.

What the *ignore-characters* list does is to specify **additional** characters to be ignored. The most likely character you want to ignore is a *digit separator*. U.S. style numbers separate digits with a comma, while European style numbers use dots for this. If your data already has separators, you can use this *ignore-characters* operand to define what that separator is. **Note**. While leading **0** and blank characters are

always ignored, the digits **1-9** are never ignored, even if you ask for this to be done. That is, digits **1-9** are always considered as significant.

Application note: If you have some unusual requirements that don't fit into what the **Z** command normally does, you have two options. First, you could try using the **RC** command to replace certain 'problem characters in the *result string* prior to the **Z** command, and then perhaps use a second **RC** command after the **Z**, to 'undo' what the first **RC** did, if that is necessary for you. If that is not sufficient, an *execute-macro string*, also known as an "**E** string", may be useful.

Minimum-length:

A decimal number that defines the right-most number of characters that will not be suppressed, regardless of their content. If *minimum-length* is omitted, there is no minimum length. That means if the original data consists entirely of blanks, zeros or other ignored characters, the end result will be a blank field of equal size. Thus, omitting the minimum length, and specifying the *minimum length* as **0** will have the same result. If the *minimum length* is equal to or greater than the current length of the *result string*, the *result string* will be left unchanged.

Example: To zero-suppress a dollar-amount field, skipping insignificant commas, and retaining a "cents portion" that already had a dot and 2 digits , use **Z','3**.

A complete mapping string that used this item would look like:

M"1+ Z','3"

or with the **1+** implied by *auto-copy* we can shorten this to:

M"Z','3"

Character-replacement item – RC

A *character-replacement item* is used to substitute one list of characters for another list of characters, wherever they may be found, on an individual basis. For example, to replace all instances of parentheses with brackets, you would replace the list of characters (and) with [and], respectively.

When a *character-replacement item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to do character replacement on the ‘final *result string* in a certain way, you would place the *character-replacement item* last (right-most) in the mapping string.

Because the action of a *character-replacement item* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of a *character-replacement item*, since the data characters in the *source string* remain undisturbed in their original locations.

Syntax of a *character-replacement item*:

RC *character-replacement-list* [*column-reference*]

RC:

Introduces the character-replacement item

Character-replacement-list:

Defines a pair of character lists. The list is a quoted *string set* or *string-pair*, in the format of:

‘before=after’ -- in string-set notation

or

‘before’ = ‘after’ -- in string-pair notation

where *before* and *after* are strings of equal length. Each character in the *result string* that are found in the *before* string is replaced, individually, by the corresponding character in the *after* string. The characters are associated by position, so the first character in *before* is converted to the first character in *after*, the second character in *before* is converted to the second character in *after*, and so on.

In this way, the *character-replacement-list* essentially defines a **character translation table**.

The *character-replacement-list* is a quoted value after the **RC** code, and inside the quotes of a *string set*, the value is always an odd length, since the *before* and *after* strings must be the same length, and the = equal sign takes up one position. When you use a *string pair*, each of the two separate strings must be the same length.

By default, the casing of *character-replacement-list* string uses the current PROFILE CASE setting in effect. This can be overridden by a **CC** or **CT** command code, or by using a type code suffix of **C**, **T** or **X** on the character-replacement-list string. When CASE T mode is in effect, by any means, the *before* string is effectively mapped twice, once for a lower-case version of itself, and once for an upper case version. For example, the command **RC'abc=123'T** maps both upper and lower case versions of its *before* string. This operates **as if** the command **RC'ABCabc=123123'C** were used. Thus, both “A” and “a” get mapped to “1”, both “B” and “b” become “2”, and both “C” and “c” become “3”.

Special considerations when using hex strings: (1) the hex value is always treated as case-sensitive, as if in **CASE C** mode; (2) the = equal sign that separates the *before* and *after* parts in a *string set* is written as

an actual = equal sign character, the same as in non-hex values. Example: RC'77=99'X.

Column-reference:

Specifies columns in the *result string* to be modified. If *column reference* is omitted, the entire *result string* can potentially be modified.

It is an error if the *character-replacement-list* in a *string set* is shorter than 3 characters, is not of odd length, or does not contain = equal sign as the center character (if written as a *string-set*). For a *string pair*, the values must be the same length and cannot be null strings.

It is illegal for an = equal sign to appear in the *before* or *after* part of a *string-set character-replacement-list*. For example, a command like `RC'<=>=[#]` is invalid.

If your *before* or *after* values need to contain = equal signs as data, the two values can be specified as a *string pair*, using the syntax `RC'before'='after'`. In this example, you could use `RC'<=>'='[#]'` instead of `RC'<=>=[#]`. When you use this *string pair* format, both the left and right strings of the pair must be the same length. This form is convenient when your strings contain equal signs as data, but can be used for any string values. When you are not dealing with embedded equal signs, the shorter form will be easier to use.

Note. If you use a *string pair*, the syntax `RC'before'='after'` allows an alternate format, as `RC'before':'after'` using a : colon instead of an = equal sign. When you do this, the comparison of the *before* value, and the substitution of the *after* value, is done in a *case-conformant* manner, as if a trailing type code of **T** appeared on the string operand.

The *character-replacement-list* represents *potential* character substitutions that may take place. It is not necessary for any of the characters in the *before* string to actually exist in the *result string*. If none are present, it is not an error, no character replacement takes place, and the *result string* will remain unmodified.

The characters in the *before* string should be unique, but this is not checked for. In case of any duplication, the *before/after* relationship of the rightmost instance is used. For example, if `RC'AAC=123'` is specified, first a relationship of $A \rightarrow 1$ is established, then that is discarded and $A \rightarrow 2$ is retained; any instances of 'A' will be replaced with '2'. The **RC** string should perform some useful work, but this is not checked for. For example, if `RC'123=123'` is specified, it is valid and will be processed, but no useful change will take place; strings will be "changed" to what they already were.

If you have many characters to replace, the *character-replacement-list* could become quite lengthy if you were doing things that involved all the letters and numbers, for instance. Keep in mind that SPFLite allows even very long primary commands to be entered, since the command line can scroll. One way to create such commands is to use SPFLite itself to build up the command on a data line, take that data line and copy it to the clipboard, then paste it into the primary command line area. You can also define commands as SET values, and then substitute their values on the command line using the `=set-name` notation.

For instance, if we wanted to implement the character replacement described above (changing parentheses into brackets), you would use a *character-replacement item* like

`RC'()=[]'`

Note that the quoted string after **RC** is of odd length (5) as there are two characters () before the = sign and two characters [] after the = sign, with the sign exactly in the middle.

A complete mapping string that used this item would look like:

`M"1+ RC'()=[]"`

or with the **1+** implied by *auto-copy* we can shorten this to:

`M"RC'()=[]"`

There is no provision for a *character-replacement item* in an **RC** command to make individual characters shorter

or longer, by deleting characters or replacing them with multi-character units. However, you could achieve the effect of that by doing it in two steps, where the first step would be to use a mapping string with a character replacement that substituted the desired characters with some special codes, and the second step would be to go back and replace the special codes with other values or with zero-length strings to delete them. You can also use the Delete Character Values command **DC** to remove specific kinds of characters from the *result string*, and you can use the String Replacement commands **RA**, **RL** and **RR** to accomplish string deletion when the *second string* operand of **RA**, **RL** or **RR** is omitted.

Suppose you wanted to find all 5-letter words, and then change all occurrences of **A**, **B** and **C** with lower case versions, but any instance of **X** you wanted removed. You could do the following, where the second step utilizes the Command Chaining technique. Here, instances of **X** are changed to the special character **^** which is then removed in the second step, which only applies that change to the lines that were altered in the first step by using the **:ZF** notation for command chaining.

```
CHANGE P'@@@@@' ALL WORD M"RC`ABCX=abc^"  
CHANGE '^" ALL :ZF
```

Likewise, if you wanted all instances of **X** to be expanded, you could use the same technique, only replacing the **^** code with some longer value.

Using the **DC** command code, you could remove the **X** in one step, as follows. Assuming that **X** is a unique value, so that you don't have to change it to some intermediate form first (like **^** above), the command would be:

```
CHANGE P'@@@@@' ALL WORD M"RC`ABC=abc` DC`X"
```

String replacement items – RA / RL / RR

A *string-replacement item* is used to replace one string with another string within the *result string*. Depending on the command code used, you can replace the left-most, right-most or all occurrences of some string with another one. If the replacement value is omitted or is a null (zero-length) string, the replacement operation acts as a deletion operation.

When a *string-replacement item* is encountered in the mapping string, it is applied to the *result string* as it exists at that point in time. Thus, to do string replacement on the ‘final *result string* in a certain way, you would place the *string-replacement item* last (right-most) in the mapping string.

Because the action of a *string-replacement item* is applied to the *result string* and not to the *source string*, the mapping string can contain additional column-reference operands to the right of a *string-replacement item*, since the data characters in the *source string* remain undisturbed in their original locations.

In the following discussion, the command **RA** will be used for illustrative purposes to generally represent the syntax of all three commands, since their format is identical. When a specific command is intended, the discussion will bring this out.

Syntax of a *string-replacement item*:

{ RA | RL | RR } *string-replacement-pair* [*column-reference*]

RA | RL | RR:

Introduces the string-replacement item.

- The command **RA** will Replace **All** occurrences of *first* within the *result string* with *second*.
- The command **RL** Replaces the **Leftmost** occurrence of *first* within the *result string* with *second*.
- The command **RR** Replaces the **Rightmost** occurrence of *first* within the *result string* with *second*.

String-replacement-pair:

Defines a pair of values. The pair is a quoted *string set* or *string-pair*, in the format of:

‘*first* =*second*’ -- in string-set notation

or

‘*first*’ = ‘*second*’ -- in string-pair notation

where *first* and *second* are text values. The *first* and *second* strings need not be of equal length. The *first* string must not be null. The *second* string can be **null**; if so, the replacement operation acts to **delete** occurrences of the *first* string.

By default, the casing of the *string-replacement-pair* uses the current **PROFILE CASE** setting in effect. This can be overridden by a **CC** or **CT** command code, or by using a type code suffix of **C**, **T** or **X** on the *string-replacement-pair*.

When **CASE T** mode is in effect, by any means, string comparisons between the *first* value and the data in the *result string* is done as case-insensitive; otherwise it is done as case-sensitive.

When you use an explicit type code of **T** on the *string-replacement-pair*, the replacement operation is performed as a *case-conformant change*. If these commands do not have an explicit type code of **T** but simply 'inherit a **PROFILE CASE T** or **CT** setting, comparison of the *first* value is done as case-insensitive but string replacement copies the *second* value as-is without attempting to be *case-conformant*. String replacement also takes place in *mode* if the separator between *first* and *second* is a **:** colon instead of an **=** equal sign.

Special considerations when using hex strings: (1) The hex value is always treated as case-sensitive, as in **CASE C** mode. (2) The **=** equal sign that separates the *first* and *second* parts in a *string set* is written as an actual **=** equal sign character, the same as in non-hex values. Example: **RA'F1F2=F8F9'X**. (3) Whether specified as a literal **=** equal sign or as a hex code, there can be only one equal sign in the hex value if specified as a single *string set*. No such restrictions exist if the value is specified as a *string pair*, like **RA'F1F2='F8F9'X**.

See the [Mapping Strings Quick Reference](#) for a concise summary of how case sensitivity and case conformance are handled.

Column-reference:

Specifies columns in the *result string* to be modified. If *column reference* is omitted, the entire *result string* can potentially be modified.

If desired, the two values can be specified as a *string pair*, using the syntax **RA'first'='second'**. This form is convenient when your strings contain equal signs as data. When you are not dealing with data having embedded equal signs, the shorter form will be easier to use. When you use the shorter *string set* form, the string must contain only one **=** equal sign.

Note. If you use a *string pair*, the syntax **RA'first'='second'** allows for an alternate format, as **RC'first':'second'** using a **:** colon instead of an **=** equal sign. When you do this, the comparison of the *first* value, and the substitution of the *second* value, is done in a *case-conformant* manner, as if a trailing type code of **T** appeared on both values.

The *string-replacement-pair* represents *potential* string replacements that may take place. It is not necessary for the *first* value to actually exist in the *result string*. If none are present, it is not an error; no string replacement takes place, and the *result string* will remain unmodified.

For instance, to replace all occurrences of one with two, you would use a *string-replacement item* like
RA'one=two'

A complete mapping string that used this item would look like:
M"1+ RA'one=two"

or with the **1+** implied by *auto-copy* we can shorten this to:
M"RA'one=two"

To **delete** all occurrences of **seven**, you would use a *string-replacement item* like:
RA'seven'

Since there is no **=** equal sign in this string operand, and it is not specified as a *string pair*, the *second* value has not been defined, and thus is null. When the commands are using in this way, the **R** in the command name can be thought of as "**Remove**" instead of "**Replace**". When *second* is omitted, **RA** = Remove All, **RL** = Remove Leftmost, and **RR** = Remove Rightmost.

In the event the value you wanted to remove has an embedded **=** equal sign, the full string-pair format is required. For example, if you wanted to remove all instances of "**A=B**" it would require a command like this:

RA'A=B='

Command-specific details:

- The **RA** command will find all occurrences of *first* and replace them with *second*, or will delete the value of *first* wherever it appears, if *second* is a null string. When **RA** finds an instance of *first* and then replaces it with *second*, as it resumes its scan of the *result string*, it will **not rescan** over any of the replacement data. Thus, if you were to specify a command of `RA'Abc=AbcAbc'`, it would **not** inspect any of the just-replaced data. The **RA** logic passes over the *second* replacement value, continuing on with its scan of the *result string* at the portion of it that had not yet been examined.
- The **RL** command will find the left-most occurrence of *first* and replace it with *second*, or will delete the value of *first* if *second* is a null string.
- The **RR** command will find the right-most occurrence of *first* and replace it with *second*, or will delete the value of *first* if *second* is a null string.

Character conversion items – AX / XA / EX / XE

A *character conversion item* is used to convert data to and from character mode and ASCII or EBCDIC encoding. This conversion is applied to a specified column range of original data. To perform this operation on the current contents of the *result string*, specify a . dot symbol instead of, or in addition to, a standard *column-range* operand.

Note that when dealing with EBCDIC conversions, special considerations apply:

1. Character conversion items in mapping strings do not know or respect the data type of the underlying file. You must know this, and apply the correct conversion for the data you are working with. That is your responsibility.
2. The facility for the **EX** and **XE** codes is described here supports EBCDIC. When you do this, SPFLite will supply a default EBCDIC translation table corresponding to code page **1140**. This is identical to the supplied EBCDIC.SOURCE code page that comes with SPFLite. There is no way to override this default.
3. You must bear in mind that, internally, SPFLite only processes data as ASCII (Ansi) characters. In an EBCDIC-encoded file, when you see a character like '1' displayed in an edit session, SPFLite sees it as an ASCII value of **X'31**, but in your data file, it exists as **X'F1**. The way editing works for EBCDIC files is that your EBCDIC data is brought into an edit session and temporarily converted to ASCII as you edit. When the file is saved, it is converted back to EBCDIC.
4. When character conversion items are introduced into this environment, if you have original EBCDIC data like **X'F1**, it is first converted to **X'31** to make it an ASCII character '1' that you recognize, when the file is opened. If you then ask for this '1' to be converted to hex using **EX**, the '1' which is internally held as **X'31** is first converted to a byte value of **X'F1**, and then that is converted to the two ASCII characters **"F1"**. Likewise, if you ask for the two characters **"F1"** to be converted using **XE**, they are first converted a single byte with the value **X'F1**. This byte value is then converted from EBCDIC to ASCII, which is **X'31**. That byte value becomes the result, which appears as the ASCII character '1' in your edit session. When your EBCDIC file is saved, that ASCII '1' will get translated (again) back to EBCDIC, resulting your file containing an EBCDIC '1' value of **X'F1**.

This may all seem rather complex, but the underlying goal is to help you “maintain the illusion” that this Ansi-based editor is really editing EBCDIC, as if you were actually running on a mainframe. SPFLite is able to maintain this illusion by very carefully translating your data at precisely the right time, and by requiring its translation tables to conform to “lossless” conversion in both directions. SPFLite’s default EBCDIC table meets that requirement.

Syntax of a *character conversion item*:

{ **AX** | **XA** | **EX** | **XE** } [*bypass-string*] *column-reference*

AX | **XA** | **EX** | **XE**:

Defines the type of character conversion to be done:

- **AX** converts individual ASCII characters (ignoring any *bypassed* characters) into two-character values, where these characters show the ASCII encoding. After conversion, the field would be twice as long as originally. For example, a string of "B 4" would become "422034", since that is what the ASCII values for these characters are.

- **XA** converts pairs of ASCII characters into single-character values. The string being so converted must contain an even number of characters, and each character may only be in the range of **0-9**, **A-F** or **a-f**. After conversion, the field would be half as long as originally. It is illegal to attempt to apply the conversion code **XA** to data which does not meet the restrictions described here. For example, a string of "422034" would become "B 4", since that is what the ASCII characters represented by these values are. **Note:** If the selected columns (ignoring any *bypassed* characters) are not spans of hex-digit characters of even length, the operation will halt with an error message.
- **EX** converts individual EBCDIC characters (ignoring any *bypassed* characters) into two-character values, where these characters show the EBCDIC encoding. After conversion, the field would be twice as long as originally. For example, a string of "B 4" would become "C240F4", since that is what the EBCDIC values for these characters are.
- **XE** converts pairs of EBCDIC characters into single-character values. The string being so converted must contain an even number of characters, and each character may only be in the range of **0-9**, **A-F** or **a-f**. After conversion, the field would be half as long as originally. It is illegal to attempt to apply the conversion code **XE** to data which does not meet the restrictions described here. For example, a string of "C240F4" would become "B 4", since that is what the EBCDIC characters represented by these values are. **Note:** If the selected columns (ignoring any *bypassed* characters) are not spans of hex-digit characters of even length, the operation will halt with an error message.

Bypass-string:

Defines an optional string containing one or more characters to be bypassed. The idea behind the *bypass string* is that you might have several spans of normal characters, or hex-digit characters, separated by blanks, commas, etc. and you want to convert the “real” data, but not the “delimiter” data. Any spans of characters contained within the *bypass string* are ignored for conversion purposes and are copied as-is to the *result string* in their relative locations, and the conversion process applies only to the remaining (non-bypassed) data. Bear in mind that, for **XA** and **XE** conversions, **each span** of characters that may be delimited by these *bypass characters* must individually be of even length and contain only hex-digit characters.

Column-reference:

Selects the columns of data in the *source string* that are to be converted (unless *dot notation* is used to select column of data from the *result string*). The syntax of this field are the same as an ordinary column reference. A *single column*, *remaining-columns*, *forward column-range* or *reverse column range* may be requested. No spaces or other characters not part of the syntax may appear between the conversion codes of **AX**, **XA**, **EX** or **XE**, and the column-reference specification.

For codes **XA** and **XE**, the column reference field must result in a selection of an even number of characters, which must consist of valid hex-digit characters. For this reason, care must be taken when using a *remaining columns* reference, such as **XA22+**. If the source data value was obtained using a Regular Expression, it is possible that the number of characters in it is variable and unpredictable, and could unexpectedly be of odd length. In those cases, it may not be possible to perform **XA** or **XE** conversions on your data. You may have to ‘normalize’ your data to ensure it only contains an even number of hex-digit characters.

For conversion codes **AX** and **EX**, when the resulting value contains hex digits greater than 9, by default, these digits will be rendered as upper-case **A-F**. If a *case conversion code* of **>** (convert to upper case) or **<** (convert to lower case) is in effect prior to the **AX** or **EX** code, those case conversions will be respected, and hex-digit characters will be rendered accordingly. Otherwise, you can use a *case alteration code* of **UC**, **LC**, **SC**, **TC** or **IC** afterwards, to alter the contents of the *result string* once it has been formed.

Numeric conversion items – DD / DX / XD / XX

A *numeric conversion item* is used to convert and reformat data between decimal and hex numeric format (using **DX** or **XD** commands), or to modify the format of a decimal or hex number without converting it (using **DD** or **XX** commands). A *decimal number* is considered to be a span of characters in the range of **0-9**; a *hex number* is considered to be a span of characters in the range of **0-9, A-F** or **a-f**. Note that numeric conversions are designed for processing integer values only; if used on a floating-point number, it may not work or may only process the part to the left of the decimal point.

When you specify a “simple” conversion item, the complete column range is expected to contain valid digits of the type indicated; otherwise an error is reported.

However, sometimes numeric data may not be in precise locations. For instance, you may have a comma-separated list of values of varying numbers of digits, and it may be necessary to “scan” your data to determine where the number is located. Your data may have leading or trailing spaces. You might need to find it using a left-to-right scan, or possibly a right-to-left scan. You may need to control how the converted number is formatted. To do these things, you can use a “formatted” conversion item.

When you use these commands, the numeric value is first converted internally to a 64-bit integer; this limits the values to 16 hex digits or 19 decimal digits. Note that no check is made for numeric overflow, so do not exceed these limits. If your command includes a *calculation-operand*, the integer value is passed to the calculator function, which can be used to calculate a derived value from your original number. Finally, this integer is converted to a decimal or hex string value.

Syntax of a *numeric conversion item*:

```
{ fixed-location-command | floating-location-command } [ format ] [ sign-code ]  
[ calculation-operand ] column-reference
```

Fixed-location-command : **DD** | **DX** | **XD** | **XX**

Defines the type of numeric conversion to be done. The columns selected may optionally contain leading and trailing spaces, and may optionally contain a leading + or – sign, but otherwise must be entirely numeric. The selected field must contain one and only one numeric value.

- **DX** converts selected columns of data from decimal characters to hex characters. When the resulting value contains hex digits greater than 9, by default, these digits will be rendered as upper-case **A-F**. If a *case conversion code* of **>** (convert to upper case) or **<** (convert to lower case) is in effect prior to the **DX** code, those case conversions will be respected, and hex-digit characters will be rendered accordingly. Otherwise, you can use a *case alteration code* of **UC**, **LC**, **SC**, **TC** or **IC** afterwards, to alter the contents of the *result string* once it has been formed.
- **XD** converts selected columns of data from hex characters to decimal characters.
- **DD** recognizes selected columns of decimal characters, allowing them to be reformatted while remaining as a decimal number.
- **XX** recognizes selected columns of hex characters, allowing them to be reformatted while remaining as a hex number.

Floating-location-command :

Floating **DD** , **DX**, **XD** and **XX** items have a special alternate syntax, as follows:

nDD		D nD
nDX		D nX
nXD		X nD
nXX		X nX

The number **n** is a *relative field location* when a numeric field is “floating” rather than being in fixed column positions. If **n** is used, it must be a single decimal digit in the range of 1-9 (a digit of 0 should not be used). See [Handling of “floating” values](#) below.

Format: [**w**] **t**

t: { **F** | **Z** | **V** }

Provides a means of describing the resulting format of the numeric value:

- **w** is a width field, specified as a numeric value of one or more decimal digits. The **w** value is optional. If omitted for type **V**, the **w** value is assumed to be 1. If omitted for types **F** and **Z**, the width of the output made the same as the input. For example, if you use a command of **DXF**, and the original decimal number had 3 digits, the hex value will have 3 digits, as if you had specified **DX3F**. A **w** value of 0 is treated the same as 1.
- **t** is a *format type code* letter. Type **F** causes the converted value to be held in a fixed number of columns, as specified by the **w** value, with non-significant positions on the left filled with “0” characters. Type **Z** causes the converted value to be held in a fixed number of columns, as specified by the **w** value, with non-significant positions on the left filled with blank characters. Type **V** causes the converted value to be rendered in a variable number of columns. For type **V**, the **w** value specifies the minimum number of digits to be rendered; if the numeric value does not need the full width **w** to hold the value, the result is padded with “0” characters sufficient to make the field at least **w** characters wide; if the value needs more than **w** characters to represent it, no padding characters are used.

Sign-code: [**S** | **C**]

If sign code **S** is present, any leading + or – sign found during processing will be retained and copied to *result string*. If sign code **S** is absent, any leading + or – signs are ignored. Any spaces after the leading + or – sign are considered to be part of the sign.

If sign code **C** is present, it is treated like sign code **S** is, with the following addition: If a number contains a **0x** or **0X** prefix (as used in C, Java, and similar languages), that prefix is considered to be part of the **sign**, and not part of a **number**. The reason code **C** is needed is that without it, a value like **0x12** might be misinterpreted to be **two** numbers, **0** and **12**, separated by a delimiter of “x”. When converting from hex to decimal and a *sign code C* is present, any **0x** or **0X** prefix is dropped. When converting from decimal to hex and a *sign code C* is present, a **0x** prefix is added.

Calculation-operand:

Allows arithmetic operations to be performed on the base numeric value before it is formatted. For a complete description of mapping string calculation operations, see [Calculation Operands](#).

For example, to perform a decimal-to-decimal reformatting with 4 fixed digits, and have the value multiplied by 2 and then having 3 added to it, you would specify a command of **DD4F'X*2+3'** or **DD4F'*2+3**.

Column-reference:

Selects the columns of data in the *source string* that are to be converted (unless *dot notation* is used to select column of data from the *result string*). The syntax of this field are the same as an ordinary column reference. A *single column*, *remaining-columns*, *forward column-range* or *reverse column range* may be requested. No spaces or other characters not part of the syntax may appear between the conversion codes of **DX**, **XD**, **DD** or **XX** (or their *formatted* counterparts), and the column-reference specification. (It is possible to select columns in such a way as to reverse the order of the original digit characters, but this would not generally be very useful.)

Commands **DX** and **XD** involve a change of radix between decimal and hex. Commands **DD** and **XX** do not involve a change of radix between decimal and hex. For all commands, decimal or hex numeric strings are scanned for and recognized, and then reformatted using the *width* and *format* types. **DD** may be thought of as “decimal-to-decimal” and **XX** as “hex-to-hex”.

Handling of “floating” values

When a *floating-location-command* with the **n** parameter is used, SPFLite will perform a “scan” of the selected value, looking for “spans” of digits. A “span” of digits is a contiguous span of characters that are valid for the type of number being converted.

The idea is that the value you are interested in is present *somewhere* in the columns of data you have selected, but the precise location where that numeric field is may vary, or “float” from one particular instance of found data to the next. That is, if you repeatedly execute your **CHANGE** command with a mapping string, between one execution of **CHANGE** to the next, the exact place where your numeric data happens to be located may vary. So, trying to “grab” such numeric data with a fixed column range won’t work.

For **DX**-type and **DD**-type conversions, spans of digit characters must be in the range **0-9**, and for **XD**-type and **XX**-type conversions, spans of digits must be in the range **0-9**, **A-F** or **a-f**. A “span” of digits is delimited by (a) the beginning and/or end of the selected value, or (b) **any character** outside the range of valid digits. After SPFLite performs such a scan, it is an error if a valid numeric value cannot be found. Bear in mind that “any” character that could act as a delimiter may be one you did not expect, such as a letter like **R**. You should understand your data to determine if these rules meet your requirements. Most numeric data you deal with will be delimited by commas or blanks, which will work fine. These commands cannot analyze your data any more precisely than determining if a character “is a digit” or “is not a digit” when looking for numbers.

You are free to scan the same original columns of data multiple times, selectively extracting more than one numeric value, by using a different **n** value to determine the relative location where the desired numeric value is to be taken from.

A “floating” command can be directed to perform its scanning process either in a **left-to-right manner**, or in a **right-to-left manner**. The way this is indicated is in the positioning of the **n** parameter.

- When the **n** value appears *to the left* of the command code, like **3DX** or **3XD**, it causes the scan to be done in a **left-to-right manner**. Here, **3DX** means to look for the third span of decimal digits (the third decimal number) going **left-to-right**. **3XD** means to look for the third span of hex digits (the third hex number) going **left-to-right**.
- When the **n** value appears *to the right of the first letter* of the command code, like **D3X** or **X3D**, it causes the scan to be done in a **right-to-left manner**. Here, **D3X** means to look for the third span of decimal digits going **right-to-left**. **X3D** means to look for the third span of hex digits going **right-to-left**.
- The same rules apply to commands of types **DD** and **XX**, when an **n** value is used, except that no radix conversion takes place.

Caution regarding the processing of floating-point (“real”) values

Be mindful that the numeric conversion items are designed to operate on integer values, **not** on floating-point (“real”) values that contain **decimal points**. For instance, if your data looks like “123.456 654.321”, and you attempt to use a “floating” version of the numeric conversion commands, this string will appear to have **four** numbers; namely, **123**, **456**, **654** and **321**. The decimal points will **not** be treated as part of the numeric values, but as **delimiters**. Using a “floating” command, the value **654** is the *third* value from the left, **not** part of a *second* value.

See the [Quick Reference](#) for examples on how to use numeric conversion items.

Sequence items – SD / SX

For some time, SPFLite users have had the ability to *enumerate* columns of data, using the enumeration keyboard functions (Enum), (EnumHexLc) and (EnumHexUc). This feature is documented in [Working with Enumerations](#) in the Help. That enumeration process involves highlighting columns of data, and then sequencing them either directly, or under the control of the Power Typing primary command **PTYPE (PT)**. That facility works fine if the sequence numbers you want to create are located in a fixed vertical column. If they are not, the enumeration keyboard functions are hard to apply to this task, or cannot be used at all.

Here are two examples where fixed enumerations are a problem:

- Example: Suppose you have a block of “sentences” or “bullet points” in a text document, and you want to number each one. You can find the first “word” in each “sentence” but if these “sentences” don’t vertically line up, how will you get the numbers on them?
- Another example: Suppose you are editing a program and you have a list of “variables”. You want to append an ascending sequence number as a suffix like `-1` or `_1` to the end of each name, but the names are of varying sizes and locations.

Sequence items enable you to perform editing tasks like this. You will note that the syntax of **SD** and **SX** is the essentially same as the *numeric conversion items* **DD**, **DX**, **XD** and **XX**, except that **SD** and **SX** do not take a *column reference* operand.

A sequence item of **SD** and **SX** is only defined when the **CHANGE** command that has your mapping string includes the keyword **ALL**. If you execute your **CHANGE** command without an **ALL** keyword, and attempt to use a sequence item of **SD** and **SX**, the value is undefined, and use of these commands without the **ALL** keyword is not supported. In some contexts, the value of the sequence number when used out of context will be zero, but this cannot be guaranteed.

When you issue a **CHANGE ALL** command with a mapping string that contains **SD** or **SX**, the first string found by **CHANGE** will have a sequence number of **1**, the second with **2**, and so on. The reason for permitting **SD** and **SX** only on **CHANGE ALL** is to ensure that the “first” found string is accurately determined, so that the choice as to which particular found string gets sequence number **1**, etc. is done correctly.

Syntax of a sequence item:

```
{ SD | SX } [ format ] [ sign-code ] [ calculation-operand ]
```

SD | **SX**:

Introduces the sequence item.

- **SD** produces a formatted sequence number in decimal
- **SX** produces a formatted sequence number in hex

Format:

Allows the resulting numeric value to be formatted. This has the same syntax and function as the *format* operand of the *numeric conversion items* **DD**, **DX**, **XD** and **XX**. Refer to those commands for information on the *format* operand.

If you issue a command such as **SDF** or **SXF** with no width field **w**, the implied number of digits (the implied **w**) will always be the same as the smallest number of digits needed to represent the value in decimal or hex, **before** any arithmetic calculations are performed by a *calculation-operand*, if you specify one. For example, suppose the value of the sequence number is **127** (hex value **7F**):

- If you specify an **SDF** command, the implied width is **3**, since the decimal value **127** has **3** digits, and the command is processed as if **SD3F** were specified.
- If you specify an **SXF** command, the implied width is **2**, since the hex value **7F** has **2** digits, and the command is processed as if **SX2F** were specified.

Since calculations could make the value larger than what is deduced for the implied width **w**, you should only rely on an implied width **w** if you are certain that no truncation will occur (or if truncation is acceptable) when you use a *calculation-operand*.

sign-code:

Manages the handing of a sign, in case the computed value is negative or if you want a “C” style **0x** prefix. This has the same basic syntax and function as the *format* operand of the *numeric conversion items* **DD**, **DX**, **XD** and **XX**, except that **SD** and **SX** never scan for a numeric value, but only produce one. Refer to the **DD**, **DX**, **XD** and **XX** commands above for information on the *format* operand. When you use a sign code of C for SX, such as SX4FC, the resultant value will have a “C”-style prefix of 0x.

Calculation-operand:

Allows arithmetic operations to be performed on the base numeric value before it is formatted. For a complete description of mapping string calculation operations, see [Calculation Operands](#).

For example, to use the sequence value with 4 fixed digits, and have the value multiplied by 2 and then having 3 added to it, you would specify a command of SD4F'S*2+3' or SD4F'*2+3'.

Specifying a calculation operand

For the sequence items **SD** and **SX**, and the numeric conversion items **DD**, **DX**, **XD** and **XX**, you are allowed to perform basic arithmetic operations on the value before it is formatted as a numeric value. This is done by specifying a *calculation operand*, which is a string operand containing an arithmetic expression.

For example, to modify a value acquired by a **DD** command, by multiplying it by 2 and adding 3, then format it as 4 digits, you would use DD4F'X*2+3' or DD4F'*2+3'.

For a complete description of mapping string calculation operations, see [Calculation Operands](#).

Exchange Items - X

An *exchange item* consists of the code **X** followed by an *exchange specification*. An exchange item is used swap two different strings at the same time. This performs a task that is difficult to do with standard CHANGE commands.

Consider the task of swapping (interchanging) two different words within some range of lines. Suppose everywhere the word “ONE” was present, you wanted “TWO”, and vice-versa (and let’s assume your data has both kinds of words). You can’t just say CHANGE ONE TWO WORD ALL, because as soon as you did, all the “ONE” words in question would become “TWO”, so you would be unable to change the *original* instances of “TWO” back to “ONE”.

Using an *intermediate value* of “///” you could try doing CHANGE ONE “///” WORD ALL first, then follow it with CHANGE TWO ONE WORD ALL, and then finally, with CHANGE “///” TWO WORD ALL. But there are problems with that. First, it takes three different commands, and they are hard to keep straight without making a mistake. Second, if your strings are WORD delimited, and your *intermediate value* is “///”, those special characters may not meet the rules you have defined for “WORD” characters.

There are also potential issues on how the change will be handled if the two strings are of different sizes, depending of whether your edit session is set for CHANGE DS mode or CHANGE CS mode; the formatting might get done improperly with all those CHANGE commands being performed with varying sizes of data, depending on what was adjacent to your data (and where) when the commands were issued.

You could possibly add “/” as a recognized WORD character on the WORD line command, but you might need your original WORD definition left alone. You could also try picking some other *intermediate value*, but you could still run into problems. Any attempt to swap these two values “the hard way” is awkward, unreliable and inefficient.

In contrast, using the *string exchange* mapping item to do this is much easier. To illustrate, exchanging the two strings we discussed above could be done with the command `CHANGE P'@@@@' M"X'ONE=TWO" WORD ALL.`

Syntax of a *string exchange item*:

X *exchange-pair* *column-reference*

X :

Introduces the string exchange item.

- **X** performs a string exchange operation. String comparisons are done in either case-sensitive or case-insensitive mode, depending on any string type code that may be present, or on the use of **CC** or **CT** command codes prior to **X**, or on the current **PROFILE CASE** setting for the file being edited.
- **X** will perform a string exchange in Text mode, if the *exchange-pair* string is specified with a trailing type code of **T**, or if the separator between the *first* string and the *second* string in a *string pair* is a : colon instead of an = equal sign. If either of these are done, string comparison is done in case-insensitive mode, regardless of any prior use of **CC** or **CT** command codes, or the current **PROFILE CASE** setting for the file being edited. Otherwise, string replacement is done as-is.

- Case-conformant changes are done comparable to the way it is done for the **CHANGE** primary command when both *string-1* and *string-2* have a type code of **T**. See the discussion of Case-Conformant Change Strings under Working with SPFLite for more information.
- See also the Mapping Strings Quick Reference for a summary of how string type codes, case sensitivity and case conformance are handled for command with string-pair operands.

Exchange-pair:

Defines the pair of strings to be exchanged. The pair is a quoted *string set* or *string-pair*, in the format of:

or

'first=second'	-- in string-set notation
'first' = 'second'	-- in string-pair notation

For example, X'one=two'. This operand can have a suffix of **C**, **T** or **X**. If the suffix is **T**, values are compared as case-insensitive. When the command has a *string-pair* separator of : colon, it implies string type **T**, and it is allowed but not necessary to also use the type code **T** when the : colon is present. To avoid ambiguity, the : colon notation is only allowed on a *string pair*, **not** in a single *string set*. That is, X'one':two' is valid but X'one:two' **is not valid**. If you want both **case-conformance and a use a string set**, you must specify it as X'one=two'T. The colon notation is just a shorthand for putting **T** on both sides of a *string pair*.

If no string suffix or colon notation is present, a **CC** or **CT** mapping item is used to specify case-sensitivity, or else the **PROFILE CASE** setting on the current edit file is used to determine case-sensitivity. In case a hex string is used to define the values of *first* and *second*, you would still use an = equal sign to separate the two values. When the command has a : colon separator like X'one':two', these considerations are ignored, because then the strings are treated as type **T** and compared as case-insensitive.

For non-hex values, in the event that the = equal sign is part of the values you are exchanging, you must specify the two values as a *string pair*. For example, if you wanted to exchange A=B with C=D, you would use M"X'A=B'='C=D'" When you are not dealing with embedded equal signs, the shorter *string set* form will be easier to use. In a *string pair*, each side of the string can use different quote types, if desired. See the Mapping Strings Quick Reference for a summary of how varying string type code combinations are handled. With a few exceptions, the various combinations of type codes work the same way they do on the *string-1* and *string-2* values of an SPFLite **CHANGE** primary command.

When the command has a : colon separator, the strings are assumed to be case insensitive, so the only use for a type code is if you wanted to specify a string in hex with an **X** code on one or both sides. Note that if you want one string to be hex and one to be non-hex, both parts of the string pair must have their own type codes; you put **X** on one side alone, an **X** is implied for the other side.

It would be unusual to treat hex value as case-insensitive. If you went to the trouble of writing out some value in hex, chances are good you wanted that specific value, and **not** some upper- or lower-case version of your hex data where some of the bytes happened to contain the hex codes for alphabetic data. Perhaps you **do** need to do this, but real uses for it will be rare.

If the mapping string has a **CT** command code prior to the **X** command, or the edit file has a **PROFILE CASE T** active, but the *string exchange item* does **not** specify a : colon separator or a string type code of **T**, string comparisons are done as case-insensitive, **without** the changes being *case conformant*.

Column-reference:

Defines a column or column-range where characters of the source data string are obtained, and checked against the *first* and *second* string values, as follows:

- If the value selected is equal to the *first* string, the value of the *second* string is appended to the end of the current *result string*.
- If the value selected is equal to the *second* string, the value of the *first* string is appended to the end of the current *result string*.
- If the value selected is not equal to either the *first* string or the *second* string, the value of the selected source data is appended to the end of the current *result string*.

Because **X** is a **copying command** that will *auto-reference*, if **X** is the first command of your mapping string, it will refer to the entire *source string*, and either store the *first* or *second* string value into the *result string*, or it will copy the value of the *source string* into the *result string*.

For example, if your mapping string is `M"X'one=two'C"` and a 3-character value is selected, then:

<i>If the value selected is ...</i>	<i>it becomes ...</i>
<code>one</code>	<code>Two</code>
<code>two</code>	<code>One</code>
<code>ten</code>	<code>Ten</code>
<code>ONE</code>	<code>ONE</code>

To perform this operation on the current contents of the *result string*, specify a `.` dot symbol instead of, or in addition to, a standard *column-range* operand. When this is done, the value obtained is appended to the current contents of the *result string*, if either the **first** or **second** string matches; otherwise the value you are testing is appended to the current contents of the *result string*.

In this example, if the *result string* originally was **one** it will now contain **onetwo**. If you really needed to create a *result string* that contains both the original value and the new value, this might be of some use. Otherwise, the `.` dot notation may not be what you need.

Here is an example of a *case conformant* exchange operation. If your mapping string is `M"X'one=two'T"` or `M"X'one': 'two"` and a 3-character value is selected, then:

<i>If the value selected is ...</i>	<i>it becomes ...</i>
<code>one</code>	<code>two</code>
<code>One</code>	<code>Two</code>
<code>ONE</code>	<code>TWO</code>
<code>two</code>	<code>one</code>
<code>Two</code>	<code>One</code>
<code>TWO</code>	<code>ONE</code>
<code>Ten</code>	<code>Ten</code>

The **X** mapping item can only exchange one pair of strings at a time. If you need capabilities beyond this, an *execute-macro string*, also known as an “**E** string”, may be useful.

Special handling when X is used on the result string

As noted, **X** is a copying command that will *auto-reference*. If you want to use **X** to exchange a string (or substring) that is **already** in the *result string*, you can use *dot notation*. For instance, if the *source string* contains “oneFiveSix” and you issued a command like `M"1-3 X'one=two'.1-3"`, the usual way commands operate is that the product of the command is **appended** to the end of the *result string*. In this example, you would first copy the value “one” and then exchange “one” and “two”, appending that to the result. The final value would have been “**onetwo**”.

For most people, that would be a peculiar result, and would not be very useful. To avoid this problem, **X** takes note that the value you asked to work on is already in the *result string*. When it is, **X will store the resulting**

value in place, and not append it. So, when the *source string* contains “oneFiveSix” and you issued a command like `M"1-3 X'one=two'.1-3"`, the *result string* will contain “two”.

Character-move item – M

You can use the Move command to move data from one location to another within *result string*, when you want to rearrange the *result string*, or when the standard default of appending values to the end of the *result string* is not what you need. The Move command, along with the Delete command **D** allow you to treat the *result string* as a sort of “scratch pad area” where you can arrange the layout of this string in ways that are more flexible than just appending values to the end, as most mapping commands do.

Syntax of a *character-move item*:

M *destination-reference* *column-reference*

M:

Introduces the character-move item.

Column-reference:

Defines the columns within the *result string* where the moved data is **coming from**. After being moved, the data in these columns is deleted. This can be any standard *column reference* operand including *end relative notation* with an ***** suffix. You cannot specify a column of **0** or **\0**. Because data always moves from one place to another within the *result string*, the *column reference* implicitly uses *dot notation* even if you don't specify an actual dot (because this is a **modifying command**). **The column reference cannot be omitted.**

Destination-reference:

A quoted string value that specifies the location where the moved data is **going to** within the *result string*. The value can be any standard *column reference* operand including *end relative notation* with an ***** asterisk suffix, or a lone ***** asterisk, in quotes. Depending on how this is specified, the move operation may involve either an insertion of data or a replacement of data. You cannot specify a column of **0** or **\0**. **The destination reference cannot be omitted.**

Because all mapping-string commands uses a fixed format syntax, which does not support having two *column reference* operands in a single command, it is necessary to put the *destination reference* into the *string operand*.

Within that string value, the *destination reference* has the same syntax as a column reference does, with the addition of a lone ***** notation:

```
first [ - second ]
| first [ - second ] *
| first / second *
| first \ second *
| *
```

In the *destination reference* string, do not include any spaces or other characters other than what is allowed by the syntax. The *destination reference* is handled as follows:

- When you specify a **single column** as **first**, the moved data is **inserted at** that point in the current *result string*, and the remaining data is **pushed** to the right. For example, if you specified a destination column of '5', the moved data is inserted **at** position 5 of the current *result string*, and whatever data was formerly located in positions 5 **and after** is pushed to the right. This means that a single column value specifies where the moved data goes before. For example, a single column of '1' means the moved-in data is inserted **before column** 1 of the existing data.
- When you specify a **column range** as **first-second**, it defines a column range within the *result string* where the data is to be **moved into**, with the former data in those columns being **deleted**. For example, if you specified a destination column range of '4-5', the moved data is inserted **into** positions 4-5 of the current *result string*, deleting whatever formerly was located in positions 4-5, and whatever data was formerly located in positions 6 **and after** is pushed to the right.
- To replace a **single column** position, you must do so with a destination column range like '5-5', because a simple column of '5' means *insertion*, not *replacement*.
- Regardless of how you specify the *destination reference* and the *column reference* values, you do so from the standpoint of how the **original source string** appears, **before** you alter it with the Move command. That is, as data is being inserted and deleted, it does not "throw off" the original column numbering of data in the *source string*.
- If you specify a column or column-range with a trailing *, like '2*' or '5-1*', it is an **end-relative** column reference, as usual. You may also specify **mixed-mode end-relative** column notation.
- If you specify the destination as a **lone '*'** it means the data is moved to the **right** of (at the **end** of) the last position in the current *result string*. This **lone '*'** feature only pertains to the destination for a Move command; it is **not** a general *column reference* notation that you could use on other mapping commands.
- If you specify a **lone '*'** to append data to the end of the *result string*, but the data you requested with the *column reference* is already located at the end of the *result string*, it is not an error, but no change will take place.
- The *destination reference* and the *column reference* columns **must not overlap**, or else an error is reported, and no modification of the *result string* will occur.
- As with other commands, if the *column reference* is outside the bounds of the actual value of the *result string*, the selected columns will be adjusted if there is at least some overlap to the actual value; if there is no overlap at all, it is considered an out-of-bounds reference, and no change will take place. For instance, if the *result string* is length 10, and you specify a *column reference* of 9-15, this will be adjusted to a range of 9-10, because that is where the *column reference* overlaps the actual data. If you specified a *column reference* of 12-15 for the same data, the Move command will be suppressed, since there is no overlap. Such a condition is not treated as an "error" but no action is taken, and the *result string* is not modified.
- Similarly, a column range destination reference is likewise adjusted to be within the bounds of the actual data. A single-column insertion-point *destination reference* is not adjusted; it must reference a valid column within the current *result string* (or be specified as '*') or else the Move command will not be performed. After these adjustments are made, there must be no overlap between the *destination reference* and the *column reference* columns.
- Because of the preceding, if you have a real need to move data to a position beyond the current length of the *result string*, you must first extend its size. That may be done with any command code that puts data in the *result string*. An obvious choice to do that would be to add blanks to the end, using the Pad Right command (PR). You can also insert literal text to extend the size of the *result string*.

As with regular *column reference* operands, the *column reference* and the *destination reference* can reference columns of data in reverse order, causing the data order to be inverted. SPFLite will not prevent you from reversing the order in either place, or even doing so in both places, which would cancel out the character

reversal. This is not considered an error, but such a double-reversal would serve no purpose. You have to understand the intent of your move operation and properly specify these operands to meet your requirements.

Example 1: Suppose your *source string* was "One\TwoXyz".

A mapping string of M" M` 5` 9-11" would do the following:

- Because Move is a *modifying command* that will *auto-copy*, the Move command will first copy the contents of the entire *source string* value of "One\TwoXyz" and place it in the *result string*. (The same thing would occur if you had starting the mapping string with a 1+ column reference.)
- The *column reference* of 9-11 would select the last 3 columns of the *result string*, which contain "Xyz". (The same columns could have been selected using *end relative notation*, via 3-1* instead.)
- Those 3 columns are saved, and (eventually) deleted from the *result string*.
- The *destination reference* of `5` is recognized as a column-insertion request. The saved columns of "Xyz" are moved into the *result string* **before** column 5. The data currently located at column 5 and beyond (which contains "\Two") is pushed to the right by 3 positions to make room for the "Xyz" value being moved in. So, the value "\Two" appears after the inserted string of "Xyz".
- To complete the move operation, the moved value "Xyz" is deleted from its old location.
- The final value of the *result string* in this example is "One/Xyz\Two".

Example 2: Again, suppose your *source string* was "One\TwoXyz".

A mapping string of M" M` 9-11` 4-5" would do the following:

- The *source string* value of "One\TwoXyz" is automatically copied to the *result string*, as in the prior example.
- The *column reference* of 4-5 would select the 2 columns in the middle of the *result string*, which contain "\".
- Those 2 columns are saved, and (eventually) deleted from the *result string*.
- The *destination reference* of `9-11` is recognized as a column-replacement request. The saved columns of "\\" are moved into the *result string* **at** columns 9-11, replacing their former contents of "Xyz".
- To complete the move operation, the moved value of "\\" is deleted from its old location.
- The final value of the *result string* in this example is "OneTwo\".

Example 3: Suppose your *source string* was "One\Two" and you wish to insert "****" in the middle, but there is no "****" currently **in** your *result string*. You can do this by inserting a "****" literal into the *result string*, and moving it to the desired location.

This procedure **cannot** rely on the *auto copy* feature of Move, since both the original *source string* and the "****" have to **already** be in the *result string* before the Move command is issued. For this process to work, we must **first** explicitly copy the *source string* into the *result string*, then insert the "****" literal, and finally issue the Move.

This is a good illustration of why the **auto copy** and **auto references** features, as convenient as they are, cannot be relied for in every situation. It's important for you to understand the "data flow" involved, to make sure you will get the results you need.

A mapping string of M"1+ `***` M`5`9-11" would do the following:

- The 1+ causes the *source string* value of "One\Two" to be copied to the *result string*.
- `***` causes this text literal to be appended; the *result string* now contains "One\Two***".
- The *column reference* of 9-11 would select the last 3 columns of the *result string*, which contain "***". (The same columns could have been selected using *end relative notation*, via 3-1* instead.)
- Those 3 columns are saved, and (eventually) deleted from the *result string*.
- The *destination reference* of `5` is recognized as a column-insertion request. The saved columns of "***" are moved into the *result string* **before** column 5. The data currently located at column 5 and beyond (which contains "\Two***" for the moment) is pushed to the right by 3 positions to make room for the "***" value being moved in. So, the value "\Two" appears after the inserted string of "***".
- To complete the move operation, the moved value is deleted from its old location.
- The final value of the *result string* in this example is "One/**\Two".

Delete character-columns item – D

You can use the Delete-characters command to delete data from specific column locations within the *result string*.

Syntax of a *delete character-columns item*:

D *column-reference*

D:

Introduces the delete character-columns item

Column-reference:

Specifies columns in the *result string* to be deleted. **The column reference cannot be omitted.** If you leave out the *column reference*, it will **not** select the entire *result string* for deletion, but will issue an error message. You are not prevented from intentionally deleting the entire *result string* by specifying an explicit *column reference* such as D1+ though generally such a command would serve no purpose.

Note that if you perform several D commands in the same mapping string, the relative column numbers to the right of where you delete data within the *result string* will change with each D command. To simplify this and avoid problems, do such multiple deletes in a right-to-left manner.

For example, suppose your *result string* contained "Abc/Def/Xyz", where the / slash characters are in positions 4 and 8. and you were trying to delete both slashes. If your mapping string contained "D4 D8", you would first delete the / slash in position 4, leaving the *result string* **at that point** with the value "AbcDef/Xyz". However, if the D8 command is then performed, the second / slash is no longer in position 8 but in 7, and the X is in 8), so that D8 will delete the X rather than the second slash, giving you a final result of "AbcDef/yz".

By switching the two commands, a mapping string of "D8 D4" will delete both slashes the right way in the right order, leaving you with "AbcDefXyz".

For similar reasons, if you were going to delete the two slashes in this same data, using **end-relative notation**, you would do so with a mapping command of "D8* D4*".

If you are interested in deleting specific *kinds* of characters, and not just at certain *locations*, you can use the Delete Character values command, described next.

Delete character-values item – DC

You can use the Delete Character values command to delete data from specific column locations within *result string*. The **DC** command has similarities to the **RC** command. The main differences are that **DC** accepts only one string value, not a *string-pair*, and the **DC** command can result in character deletions, while the **RC** command does individual, one-for-one character replacements.

Syntax of a *delete character-values item*:

DC *character-deletion-list* [*column-reference*]

DC:

Introduces the delete-character-values item

Character-deletion-list:

Defines a string of one or more characters.

Each character in the *result string* that are found in the *character-deletion-list* string is deleted.

By default, the casing of *character-deletion-list* string uses the current **PROFILE CASE** setting in effect. This can be overridden by a **CC** or **CT** command code, or by using a type code suffix of **C**, **T** or **X** on the character-replacement-list string. When **CASE T** mode is in effect, by any means, any **letters** in the *character-deletion-list* string are effectively recorded twice, once for the lower-case version of the letter, and once for the upper case version. For example, the command **DC'abc'T** maps both upper and lower case versions of the *character-deletion-list* string. This operates as if the command **DC'ABCabc'C** were used. Thus, it will delete both upper-case and lower-case versions of A, B and C.

When using hex strings, the hex value is always treated as case-sensitive, if any of its data contains letters. So, a command of **DC'61'X** would delete the lower-case letter 'a' but not the upper case letter 'A'.

Column-reference:

Specifies columns in the *result string* where characters can potentially be deleted. If *column reference* is omitted, the entire *result string* can potentially be modified.

The *character-deletion-list* represents *potential* character deletions that may take place. It is not necessary for any of the characters in the *character-deletion-list* string to actually exist in the *result string*. If none are present, it is not an error; no character deletions takes place, and the *result string* will remain unmodified.

The characters in the *character-deletion-list* string should be unique, but this is not checked for. Any duplications are simply ignored.

For instance, to delete all slash characters from the string "Abc/Xyz/Pdq", you could use a *delete character-values* like

DC'/'

A complete mapping string that used this item would look like:

M"1+ DC'/"

or with the 1+ implied by *auto-copy* we can shorten this to:

M"DC'/"

The final *result string* will be "AbcXyzPdq".

Mapping String Calculation Operands

Topics

[Specifying a calculation operand](#)

[Introduction to the calculator](#)

[Specifying numeric literals](#)

[Specifying large numeric literals](#)

[Expression Operators](#)

[Operation and use of calculations in mapping strings](#)

[Calculator command-line test program](#)

Specifying a calculation operand

For the sequence items **SD** and **SX**, and the numeric conversion items **DD**, **DX**, **XD** and **XX**, you are allowed to perform arithmetic operations on the value before it is formatted as a numeric value. This is done by specifying a *calculation operand*, which is a string operand containing an arithmetic expression string. The expression string is processed by a *calculation function*, which evaluates the expression and produces a numeric result, which in turn is formatted by the **SD**, **SX**, **DD**, **DX**, **XD** and **XX** mapping items.

You can invoke an SPFLite **primary command** from an SPFLite macro, using the built-in macro function **SPF_Cmd()**. When you do this, and your SPFLite command is a CHANGE command with a **mapping string** (a **string-2** operand with an **M** string), the mapping string can contain one or more **SD**, **SX**, **DD**, **DX**, **XD** or **XX** mapping items, which in turn may have *calculation operands*.

You can also invoke the *calculation function* directly from an edit macro, using the built-in function **SPF_Calc()**. See the macro documentation for more information, by issuing the command **HELP MACRO** from the SPFLite primary command line.

You are likely to find the *calculation function* will allow you to perform any calculations you might need in connection with your use of mapping strings, and then some. The design of the calculation features was made with future enhancements and developments of SPFLite in mind.

Introduction to the calculator

The *calculation function* has the following features.

1. The *calculation function* accepts a *calculation operand* as a string value, containing one or more arithmetic expressions. It processes this string and produces a numeric result. Only integer calculations are performed. The string is treated as case-insensitive.
2. The *calculation operand* may contain one or more expressions, separated by ; semicolons. The main reason to specify more than one expression is to perform assignments to more than one variable, as generally only one variable is assigned per expression. In this sense, “expressions” are somewhat like “statements” in a programming language, although as used in the *calculation function*, it is not quite so formal. The last or only expression may optionally omit the final ; semicolon. Extra semicolons are ignored.
3. The *calculation function* operates with 64-bit integer arithmetic, which has a capacity of 19 decimal digits or 16 hex digits. The highest decimal value is 9,223,372,036,854,775,807 and so there are actually 18 full

decimal digits available. Note that no check is made for numeric overflow. For applications involving the editor, it is unlikely you would exceed this limit.

4. When the *calculation function* is called, it is provided with an *initial value*. For the numeric conversion items **DD**, **DX**, **XD** and **XX**, the *initial value* is the number acquired by the mapping item, and for the sequence items **SD** and **SX**, the *initial value* is the *sequence number*. When you invoke the *calculation function* from an edit macro using the built-in function **SPF_Calc()**, the *initial value* is one of the supplied arguments.
5. The *calculation function* provides 26 *variables* named A through Z, which you can use like variables in a conventional program. These variables are 64-bit integer values, which can be read and modified. A variable is read by its appearance as a term in an expression, and is modified by being on the left side of an assignment or swap operator. The variable names are case insensitive.
6. When the *calculation function* begins, the variables are pre-initialized as follows:
 - Variable **X** and variable **R** both contain the *initial value*.
 - Variable **s** contains the contents of the *sequence number*.
 - Variable **I** contains the line number where the **CHANGE** command found the *source string*.
 - Variable **C** contains the beginning column number where the **CHANGE** command found the *source string*.
 - For the **SD** and **SX** commands, variables **x**, **R** and **s** will all have the same value.
 - The remaining 21 variables will be initialized to zero.
 - You are free to modify any variables. Because the pre-initialized variables noted above define initial conditions, you may wish to treat these variables as read-only and avoid modifying them, unless their pre-initialized values are not needed.
 - When you call the *calculation function* from the SPFLite built-in function **SPF_Calc()**, the pre-initialization of variables is handled somewhat differently. See the Help entry for **SPF_Calc()** in **HELP MACRO** for more information.
7. For variable **s**, the *sequence number* value associated with it is the *ordinal number* of the order in which a string was found during a **CHANGE ALL** command. During the course of a **CHANGE ALL** operation that contains a mapping string, which in turn contains a calculation operand, the *first* string so found has a sequence number of 1, the *second* has a sequence number of 2, etc. You can use this information to tailor the way you calculate the value returned from your expression. For instance, you may wish to develop a sequence number as a prefix or suffix of some name or string, or, you may wish to enumerate paragraphs or sections of a document that are not amenable to the use of the Enumeration keyboard functions (which require aligned columns of numbers). If you use the *calculation function* in a context for which there is no defined sequence number, variable **s** will contain zero.
8. After you perform your calculations, the *calculation function* determines and returns a final value as follows:
 - If you have assigned a new value to variable **R**, the last value assigned to variable **R** will be used as the *returned value*. Variable **R** takes on a role that a **RETURN** statement or a *function result value* provides in many programming languages.
 - If you have **not** assigned a new value to variable **R**, but you **have** assigned a new value to variable **x**, the last value assigned to variable **x** will be used as the *returned value*.
 - If you do not assign a value to either variable **R** or variable **x**, but have specified one or more expressions, the value of the right-most or only expression is assumed to be the *returned value*. Because of such assumed returned values, you can make the expression simple and brief by omitting any assignments, if these are not needed. For example, if you just want to add 5 to the initial value, this may be done with **x+5**. When 5 is added to the initial value, it leaves the result as the last-evaluated expression. Since the expression **x+5** makes no assignments to **R** or **x**, it is assumed that its calculated value is what you want as the result. For purposes of determining a *returned value*, assignments to any variables other than **R** and **x** are ignored.
 - If you have both assignments and references to variable **R** or variable **x**, only the assignments will

be considered when determining the return value. For example, if the initial value is **10**, the expression **R=X*2 ; R+1** will return **20** rather than **21**. That is because the last assignment to **R** set it to **20**, and the fact that the *last expression* had a value of **21** is disregarded. If you wanted that final value to be returned, you would either have to make a second assignment like **R=X*2 ; R+=1** or you could omit all assignments and have a simple expression like **X*2+1**.

Because the *calculation function* will often be used for very simple calculations, a further expression simplification is supported. When the leftmost part of the *calculation operand* is a **binary operator**, it is assumed that the expression is preceded by **an implied X variable**. When the *calculation operand* begins with + or – these are assumed to be binary operators (not unary), with an implied **X** on the left. For instance, a *calculation operand* of **+5** will be assumed to mean **X+5**, and then this will be treated as described in the preceding paragraph

When you use this shortcut, you must ensure that the resulting expression, **including the implied X variable**, is syntactically correct.

The **X** variable is **not** implied when the *calculation operand* has a leading operator that is only used as unary. For example, a calculation operand of **!4** would not imply **X!4** because **!** is the unary NOT operator. Thus, **!4** would calculate the value of **!4** without reference to variable **X**.

The only case where this could be an issue is if you needed a leading + or – unary operator, since the **implied X** rules would be used, and a calculation adding to or subtracting from **X** would be performed. An actual unary + operator is never algebraically required, and can be omitted. However, if you needed an expression like **-4*S** and did **not** want this to mean, **X-4*S**, you would need to change your *calculation operand*. For the unary – operator, if you really need to begin your expression this way, you can do any of the following:

- Precede the initial unary – operator with a ; , semicolon.
example: **; -4*S**
- Precede the initial unary – operator with a **0**, which is numerically the same.
example: **0 -4*S**
- Enclose the term having the initial unary – operator in parentheses.
example: **(-4)*S** or **(-4*S)**
- Rewrite the expression so it does not begin with an initial unary – operator.
example: **S*-4** or **4*-S**
- Rewrite the expression so it explicitly assigns variable **R** or **X**.
example: **R=-4*S**

- o If you specify a blank or empty expression, it will be determined that there **is** no final value and that neither variable **R** nor variable **X** have been assigned, and then the *returned value* will be made the same as the *initial value*. You can use this fact to specify blanks as a “no-operation” expression, if you sometimes don’t want to alter the initial value. Other “no-operation” expressions that will return the initial value unchanged include **R**, **X**, **R=X** and expressions that only contain semicolons.
- o The *calculation function* uses an expression syntax largely modeled on the C programming language. This was done to make these expressions as concise as possible. Some extensions to C syntax were made, to make it compatible with common usage in other languages, and to add additional operators to take the place of common arithmetic functions.
- o A large number of arithmetic operators are available to calculate numeric values. These are unary (prefix) and binary (infix) operators. A number of binary operators can be used as *arithmetic assignment operators*. For example, to multiply variable **A** by **5**, you can use **A = A * 5**, or as **A *= 5** with the ***=**

arithmetic assignment operator. Unlike the C language, there are no increment or decrement operators like `++` and `--`. If you want to increment variable `A`, you must use `A = A + 1` or `A += 1` rather than `A++` or `++A`. As noted above, if you just want to increment or decrement the initial value to produce the result value, you can do this with a *calculation operand* of `+1` or `-1` respectively.

- When an operator like `*` can be used as an *arithmetic assignment operator*, it is said to be **assignable**. Any binary operator that is *assignable* may be used to modify a variable by putting a variable name on the left, then the operator, then an `=` equal sign. Thus, the *assignable* multiply operator `*` becomes `*=` with a variable name on its left, and an expression on its right, as in `A *= 5`. Because many operators are assignable, this is simply noted in the table, rather than listing them all separately. If you are familiar with C expressions, assignable operators like `A += 1` should be clear. Otherwise, you can use the longer form `A = A + 1` if you prefer.
- Calculation expressions follow typical rules for expression formation. Operator precedence, associativity and nested parentheses are handled as usual. An operator precedence table appears below. The table largely follows C expression rules.
- Numeric literals may be specified in decimal or hex. In either case, these values are converted to 64-bit integer values. An extended notation is available for specifying large numeric values more concisely.
- Variable names and/or numeric literals may not be adjacent to each other, either with or without intervening blanks. Some kind of operator, parentheses or other punctuation must separate these items. You can use underscores to clarify large numbers, such as `123_456`. You cannot begin a number with an underscore but these can be freely used elsewhere within the number. Blanks in expressions are ignored, except that multiple-character operators like `<>` and numeric literals must not have embedded blanks.
- There is no direct support for conditional calculations, such as the `?` and `:` conditional operators in C. There is also no facility for defining tables or doing table lookups. Instead, the *calculation function* uses the concept of *set expressions* and *set indexing* to serve both of these purposes, using a simplified syntax.

Specifying numeric literals

Literals may be specified in decimal or hex. You can optionally use underscores for readability with large numbers, and these will be ignored.

- For decimal numbers, up to 19 digits may be used, and 18 *full* decimal digits are available. The largest positive value is 9,223,372,036,854,775,807, and the smallest negative number is –9,223,372,036,854,775,808. If you attempt to specify a number outside these limits, an overflow will occur, and the resulting value will be the smallest negative number.
- For hex numbers, up to 16 hex digits may be used. Hex numbers are introduced by a leading `.` dot. Hex digits > 9 are A to F and are case insensitive. For example, `255` and `.FF` are the same value, and so are `65_535` and `.F_FFF` .

Specifying large numeric literals

The *calculation function* provides a means to concisely specify large decimal and hex values that have repeating digits. It is somewhat similar to E notation commonly used for floating point numbers, but the same syntax provided will work in both decimal and hex integers (the E notation would not work in hex, since E is a hex digit) and is more flexible.

Large numbers may be required in applications involving calculations of checksums, hash keys or encryption values on your user data.

If your work does not involve large numbers, you can skip the rest of this section.

The large number syntax relies on three suffixes, comparable to floating-point **E** notation, namely a “Low Suffix”, a “High Suffix” and a “Repeated Suffix”. These are used the same as you might specify a floating point value of 5E4 to mean 50000.0. A number may have only one suffix, which appears at the end of the number without any intervening blanks. These suffixes operate as follows:

- A *Low Suffix* consists of the letter **L** followed by a decimal number. The Low Suffix indicates the number of **0** digits that are to be appended to the number. For example, to concisely specify the decimal number **700_000**, you would use **7L5**. To concisely specify the hex number **.F00_000**, you would use **.FL5**.
- A *High Suffix* consists of the letter **H** followed by a decimal number. The High Suffix indicates, **for decimal numbers**, the number of **9** digits that are to be appended to the number, and **for hex numbers**, the number of **F** digits that are to be appended to the number. For example, to concisely specify the decimal number **799_999**, you would use **7H5**. To concisely specify the hex number **.7FF_FFF**, you would use **.7H5**
- A decimal number requires one leading digit before specifying a suffix. For instance, to concisely specify the decimal number **999_999_999**, you would could use either **9H8** (an initial 9 digit, plus eight more 9s) or **0H9** (an initial 0 digit which is numerically ignored, plus nine more 9s)
- Because a hex number is already introduced by the leading dot, it is not necessary to have any leading digits before specifying the hex High Suffix. For instance, to concisely specify the hex number **.FFFF_FFFF**, you could use **.H8**.
- A *Repeat Suffix* consists of a decimal or hex digit, then the letter **R** followed by a decimal number. The Repeat Suffix indicates the number of specified digits that are to be appended to the number, in cases where the digit in question being repeated is not necessarily **0**, **9** (for decimal) or **F** (for hex) where the **L** or **H** suffix might have been used. Because of the need to specify a digit to be repeated, you number must use at least one leading digit before using this suffix. For example, to concisely specify the decimal number **733_333**, you would use **73R5**. To concisely specify the hex number **.F33_333**, you would use **.F3R5**. Note that the count after the **R** reflects the total number of digits, including the digit that precedes the **R**. For this reason, a repetition number less than 3 will not reduce the size of the literal and would normally not be used.

The decimal number in a suffix is limited to 2 digits. Its value may not exceed 19 for a decimal number or 16 for a hex number; the minimum value is 1 in both cases. Additionally, the suffix cannot cause the final number of digits to exceed 19 for a decimal number or 16 for a hex number. If the decimal number in a suffix is outside these limits, a syntax error is detected and the calculation halts. For the mapping string, the returned value will be unchanged from the initial value. Readability underscores in numbers are ignored and do not affect the maximum number of digits allowed.

Expression Operators

The following table describes the operators, their precedence and associativity. Operators with higher precedence are performed ones with lower precedence. Higher-precedence operators are listed with higher precedence numbers. Operators at level 15, if present, are performed first, while operators at level 1, if present, are performed last.

This precedence follows the traditional C language expression syntax and semantics. As is true of many beginning C programmers, some may find the extensive number of operators and precedence levels to be intimidating. However, experience has shown that this arrangement actually implements the most frequently desired choice of actions, in cases where specifying the fewest number of parentheses is desired. Of course,

whenever in doubt, you are free to use parentheses as needed to clarify to the *calculation function* and to yourself what your intentions are.

Headings:

- Oper: Operator
- Prec: Precedence level
- Assoc: Associativity:

LTR = Left to Right

RTL = Right to Left

NON = Non-Associative. Some operators are non-associative, because allowing two or more such operators in a row (without using parentheses) may be ambiguous or confusing.

Note: Even where operators may be LTR or RTL, expressions must still be written carefully and properly. For example, $R = x < y < z$ is a “valid” expression, but it probably will not do what you want. First, $X < Y$ is evaluated; if true, its value 1, else 0. Then, Z is compared to **that** result, so that you would be either be comparing $0 < Z$ or $1 < Z$. If your intent was to determine if X , Y and Z were in numeric order, the right way to do that is with $R = (x < y) \&& (y < z)$.

- Asgn: Assignable. If an operator is assignable, you can make the operator an assignment operator by adding a trailing = equal sign.. For instance, the multiply operator * is assignable, so you can use a *= multiply-and-assign operator. Only binary operators are capable of being assignable. Operators for which being assignable is not applicable will appear as “n/a”. When assignability is shown as No, it reflects an operation seldom needed or not easily incorporated by the syntax. If needed, the effect can usually be accomplished by a longer expression with an explicit assignment operator.
- When you use an assignment operator such as *= it no longer has its original precedence (the * operator has an original precedence of 14) but rather the same precedence as the := assignment operator; namely, 2. All such assignment operators like *= associate Right to Left, even though * associates Left to Right. (Associating Right to Left is necessary for assignments, because the expression on the right has to be evaluated first, before it can be assigned to the variable on the left.)

Oper	Prec	Assoc	Description	Asgn
+	15	RTL	Unary arithmetic positive, ignored except to clarify an expression. Not needed, but supported to conform to conventional usage.	n/a
-	15	RTL	Unary arithmetic negation	n/a
!	15	RTL	Unary logical not If $x = 0$, $!x = 1$ If $x \neq 0$, $!x = 0$	n/a
~	15	RTL	Unary bitwise negation (one's complement)	n/a
@	15	RTL	Unary absolute value, like ABS function in Basic If $x < 0$, $@x = -x$ If $x \geq 0$, $@x = x$	n/a
\$	15	RTL	Unary sign function, like SGN function in Basic If $x < 0$, $$x = -1$ If $x = 0$, $$x = 0$ If $x > 0$, $$x = 1$	n/a
\$\$	15	RTL	Unary sign index function, like 2+SGN function in Basic If $x < 0$, $$$x = 1$ If $x = 0$, $$$x = 2$ If $x > 0$, $$$x = 3$	n/a

*	14	LTR	Multiply	
/	14	LTR	Integer divide	
% \	14 14	LTR LTR	<p>Modulus (remainder) <code>%</code> is the C language operator, <code>\</code> is a common alternative notation. Both have same meaning and are interchangeable.</p> <p>Note that Integer Divide and Modulus (remainder) operators calculate in such a way that the quotient and remainder will have the same absolute values regardless of the signs of the dividend and divisor. This is done by doing a preliminary Divide or Modulus using the absolute values of the numeric terms, and then correcting the signs afterwards. Attempting a Divide or Modulus with a right-hand term of zero is an error which will halt the calculation and the mapping string that contains it.</p>	Yes
+	13	LTR	Add	Yes
-	13	LTR	Subtract	Yes
<<	12	LTR	<p>Bitwise (unsigned) shift left. Zero bits are filled on the right.</p> <p>If you shift with a negative shift amount, it will reverse the direction of the shift. An expression of <code>x << 4</code> will shift left 4 bits, while an expression of <code>x << -4</code> will operate the same as <code>x >> 4</code>. The determination of a negative shift amount is not just a matter of an explicit – minus sign operator, but is based on the evaluation of the expression.</p>	Yes
>>	12	LTR	<p>Bitwise (unsigned) shift right. Zero bits are propagated on the left.</p> <p>If you shift with a negative shift amount, it will reverse the direction of the shift. An expression of <code>x >> 4</code> will shift right 4 bits, while an expression of <code>x >> -4</code> will operate the same as <code>x << 4</code>. The determination of a negative shift amount is not just a matter of an explicit – minus sign operator, but is based on the evaluation of the expression.</p>	Yes
+>	12	LTR	<p>Numeric (signed) shift right. The sign bit is propagated on the left.</p> <p>If you shift with a negative shift amount, it will reverse the direction of the shift. An expression of <code>x +> 4</code> will shift right 4 bits, while an expression of <code>x +> -4</code> will operate the same as <code>x << 4</code>. The determination of a negative shift amount is not just a matter of an explicit – minus sign operator, but is based on the evaluation of the expression.</p> <p>The operator <code>+></code> is equivalent to the <code>>>></code> operator in Java.</p>	Yes
#	12	LTR	Numeric scale	Yes

			<p>In the expression x # s, the value X is divided by S. Each division operation is counted, with the first one treated as division number 1. After division, a test is made to see if the result is 0; if so, the # operator halts; otherwise, the quotient is divided again and process repeats. The result of the # operator is the count of the number of divisions performed. If S < 2, x # s produces 0.</p> <p>When S is 10, x # s is the number of decimal digits required to represent X. For example, 123456 # 10 will produce 6.</p> <p>When X is zero and S >= 2, x # s will produce 1.</p>	
##	12	LTR	<p>Bitwise scale</p> <p>In the expression x ## s, the value X is unsigned-shifted right by s, the same as done by x >> s. Each shift operation is counted, with the first one treated as shift number 1. After a shift, a test is made to see if the result is 0; if so, the ## operator halts; otherwise, the result is shifted again and process repeats. The result of the ## operator is the count of the number of shifts performed.</p> <p>If S < 1 or S > 63, x ## s produces 0.</p> <p>When S is 4, x ## s will report the number of hex digits required to represent X. For example, .FF123 ## 4 will produce 5. When S is 1, x ## s is the number of binary digits (bits) required to represent X.</p> <p>When X is zero, and 1<=S<=63, x ## s will produce 1.</p>	Yes
>	11	LTR	<p>Greater Than.</p> <p>If X is > Y, the value of x > y is 1. If X is not > Y, the value of x > y is 0.</p>	No
<	11	LTR	<p>Less Than.</p> <p>If X is < Y, the value of x < y is 1. If X is not < Y, the value of x < y is 0.</p>	No
>=	11	LTR	<p>Greater Than or Equal To.</p> <p>If X is >= Y, the value of x >= y is 1. If X is not >= Y, the value of x >= y is 0.</p>	No
<=	11	LTR	<p>Less Than or Equal To.</p> <p>If X is <= Y, the value of x <= y is 1. If X is not <= Y, the value of x <= y is 0.</p>	No

!>	11	LTR	<p>Not Greater Than. This operator is equivalent to a MIN function in some programming languages.</p> <p>In the expression $x \ !> y$, the intent is to limit the value of X such that it is not greater than Y.</p> <p>If X is $> Y$, the value of $x \ !> y$ is Y. If X is not $> Y$, the value of $x \ !> y$ is X.</p>	No
!<	11	LTR	<p>Not Less Than. This operator is equivalent to a MAX function in some programming languages.</p> <p>In the expression $x \ !< y$, the intent is to limit the value of X such that it is not less than Y.</p> <p>If X is $< Y$, the value of $x \ !< y$ is Y. If X is not $< Y$, the value of $x \ !< y$ is X.</p>	No
!>=	11	LTR	<p>Not Greater Than or Equal To</p> <p>In the expression $x \ !>= y$, the intent is to limit the value of X such that it is not greater than or equal to Y.</p> <p>If X is $>= Y$, the value of $x \ !>= y$ is Y-1. If X is not $>= Y$, the value of $x \ !>= y$ is X.</p>	No
!<=	11	LTR	<p>Not Less Than or Equal To</p> <p>In the expression $x \ !<= y$, the intent is to limit the value of X such that it is not less than or equal to Y.</p> <p>If X is $<= Y$, the value of $x \ !<= y$ is Y+1. If X is not $<= Y$, the value of $x \ !<= y$ is X.</p>	No
!= <>	10 10	LTR LTR	<p>Not Equal To</p> <p>!= is the C language operator, <> is a common alternative notation. Both have same meaning and are interchangeable.</p> <p>If X is equal to Y, the value of $x \ <> y$ is 0. If X is not equal Y, the value of $x \ <> y$ is 1.</p>	No
==	10	LTR	<p>Equal To. == is the C language operator.</p> <p>If X is not equal to Y, the value of $x \ == y$ is 0. If X is equal Y, the value of $x \ == y$ is 1.</p>	No
=	10	LTR	<p>Equal To. The <i>calculation function</i> defines the = operator by context. When it appears enclosed within () parentheses, the = operator is treated as an alias for the == comparison operator, meaning it has the same priority and associativity.</p> <p>When = appears with [] set enclosures or is not enclosed, it is treated the same as the := explicit</p>	No

			<p>assignment operator. Only one <i>unenclosed =</i> may appear in a given <i>statement</i> or <i>set clause</i>. This means that expressions like <code>x = y = z</code> are illegal.</p> <p>The <i>calculation function</i> performs this context determination in order to prevent common misuse of <code>=</code> and <code>==</code> inside of comparison sub-expressions, to help prevent errors.</p> <p>Because a plain <code>=</code> outside of parentheses is assumed to be an assignment, if you wanted to perform a comparison instead, you must either use an explicit <code>==</code> comparison operator, or put the expression with <code>=</code> in parentheses.</p>	
<code>&</code>	9	LTR	Bitwise AND	Yes
<code>^</code>	8	LTR	Bitwise XOR	Yes
<code> </code>	7	LTR	Bitwise OR	Yes
<code>&&</code>	6	LTR	<p>Logical AND. <code>x && y</code> is equivalent to <code>!!x & !!y</code>.</p> <p>If <code>X=0 or Y=0, x && y = 0</code> If <code>X not = 0 and Y not = 0, x && y = 1</code></p>	Yes
<code>^^</code>	5	LTR	<p>Logical XOR. <code>x ^^ y</code> is equivalent to <code>!!x ^ !!y</code>. The operator returns 1 if X and Y are logically different.</p> <p>If <code>X=0 and Y=0, x ^^ y = 0</code> If <code>X = 0 and Y not = 0, x ^^ y = 1</code> If <code>X not = 0 and Y = 0, x ^^ y = 1</code> If <code>X not = 0 and Y not = 0, x ^^ y = 0</code></p>	Yes
<code> </code>	4	LTR	<p>Logical OR. <code>x y</code> is equivalent to <code>!!x !!y</code>.</p> <p>If <code>X=0 and Y=0, x y = 0</code> If <code>X not = 0 or Y not = 0, x y = 1</code></p>	Yes
<code>::</code>	3	NON	<p>Swap variables. Both the left-hand operand and the right-hand operand must be variables. The contents of the two variables are swapped.</p> <p>It is not an error if the two variables are the same, but this would serve no purpose.</p> <p>Once the operator is executed, the value of the expression as a whole is the value of the variable on the left side after the swap occurs. That is, after <code>X :: Y</code> is performed, the value of the expression will be the new value of X which will be the former value of Y.</p>	No
<code><:</code>	3	NON	Conditional swap variables for <code><=</code> Both the left-hand operand and the right-hand	No

			<p>operand must be variables. The operator implements a “sorting” or “ordering” function. In the expression <code>x <: y</code>, the intent is to conditionally swap the contents of the two variables, if necessary, so that the relationship $X \leq Y$ holds true. If $X \leq Y$ is already true, no swap occurs. If $X \leq Y$ is false, a swap occurs.</p> <p>It is not an error if the two variables are the same, but this would serve no purpose.</p> <p>Once the operator is executed, the value of the expression as a whole is the value of the variable on the left side, whether swapped or not. That is, after <code>X <: Y</code> is performed, the value of the expression will be the new value of X which will be the former value of Y if a swap occurred, otherwise it will be the former value of X.</p>	
<code>>:</code>	3	NON	<p>Conditional swap variables for <code>>=</code></p> <p>Both the left-hand operand and the right-hand operand must be variables. The operator implements a “sorting” or “ordering” function. In the expression <code>x >: y</code>, the intent is to conditionally swap the contents of the two variables, if necessary, so that the relationship $X \geq Y$ holds true. If $X \geq Y$ is already true, no swap occurs. If $X \geq Y$ is false, a swap occurs.</p> <p>It is not an error if the two variables are the same, but this would serve no purpose.</p> <p>Once the operator is executed, the value of the expression as a whole is the value of the variable on the left side, whether swapped or not. That is, after <code>X >: Y</code> is performed, the value of the expression will be the new value of X which will be the former value of Y if a swap occurred, otherwise it will be the former value of X.</p>	No
<code>[]</code>	2	RTL	<p>Set expression</p> <p>A set expression provides a means both to evaluate expressions conditionally, and to do a simple table-lookup operation.</p> <p>The general form of a set expression is:</p> $\text{expr select [expr}_1\text{, expr}_2\text{, ... expr}_n\text{]}$ <p>Each expression inside the brackets is called a <i>set clause</i>.</p> <p>The <i>selector expression</i> is evaluated.</p> <ul style="list-style-type: none"> • If the value of the <i>selector expression</i> <code>expr select</code> is 	n/a

- 1, then **expr₁** becomes the value of the *set clause*.
- If the value of the *selector expression* **expr_{select}** is 2, then **expr₂** becomes the value of the *set clause*, etc.
 - If the value of the *selector expression* **expr_{select}** is < 1 or > **n** then **expr_n** becomes the value of the *set clause*.

If your *selector expression* is a conditional operator, in which *true* expressions evaluate to 1 and *false* expressions evaluate to 0, you would write an expression that used such a condition like this:

conditionalExpression [*truePart* ,
falsePart]

For example, if you wanted a value of 123 if X is greater than 5, otherwise a value of 456, you would use:

X > 5 [123, 456]

Here, if X>5 is true, the condition evaluates to 1, and the first set clause (123) is selected. If X>5 is false, the condition evaluates to 0, which is an *out-of-bounds set selector*, resulting in the last set clause of the set expression (456) being selected by default.

Suppose you wanted to set the *return* variable **R** (or other variable) to one of these two values. In a programming language, you might do this with a statement something like,

**if x > 5 then r = 123 else r = 456
endif**

In a calculation operand, you could do the same thing with:

R = X > 5 [123, 456]

You are allowed to use assignment operators within a set expression. For example, if you wanted a value of 123 assigned to variable R if X is greater than 5, otherwise a value of 456, you can rewrite the expression above as:

X > 5 [R=123, R=456]

When you use such expressions, only the selected set clause is executed, including any variable assignments within it; the remainder is skipped.

			<p>Variable assignment operators in set clauses that are not selected are not performed.</p> <p>If you want to use a <i>set expression</i> to look up values, you write the set clauses in the order you want the selector expression to choose them. Suppose variable X was supposed to be from 1 to 3, and you wanted this converted to some special values, but set to 0 if it was out of range. You could do that with a set expression like this:</p> <p style="text-align: center;">X[123, 456, 789, 0]</p>	
=	1	NON	<p>Assignment. The <i>calculation function</i> defines the <code>=</code> operator by context. When it appears within parentheses, the <code>=</code> operator is treated as an alias for the <code>==</code> comparison operator. When <code>=</code> appears with <code>[]</code> set enclosures or not enclosed, it is treated the same as the <code>:=</code> operator. Only one <i>unenclosed =</i> may appear in a given <i>statement</i> or <i>set clause</i>.</p> <p>This means that expressions like <code>x = y = z</code> are illegal, because when the <code>=</code> operator is considered an assignment operator, it is <i>non-associative</i>. That means the second equal sign has neither higher nor lower precedence than the first one, but is simply illegal. The first <code>=</code> equal sign between <code>X</code> and <code>Y</code> is valid, but the one between <code>Y</code> and <code>Z</code> is invalid. The reason this is done is to prevent confusing the intent of the expression.</p> <p>If the intent is to compare <code>Y</code> and <code>Z</code> for equality, and assign the comparison result to <code>X</code>, use <code>x = (y = z)</code> or <code>x = y == z</code>.</p> <p>If the intent is to first assign <code>Z</code> to <code>Y</code>, then assign <code>Y</code> to <code>X</code>, use <code>x = y := z</code> or <code>x := y := z</code>.</p> <p>The <i>calculation function</i> performs this context determination in order to prevent common misuse of <code>=</code> and <code>==</code> inside of comparison sub-expressions.</p> <p>The left-hand operand must be a variable name. The value of an assignment expression is the value assigned to the variable.</p>	n/a
<code>:=</code>	1	RTL	<p>Assignment. The left-hand operand must be a variable name. The value of an assignment expression is the value assigned to the variable.</p> <p>The <code>:=</code> operator may be used in chained assignments (as shown above) and inside of parentheses and set expressions.</p>	n/a

Operation and use of calculations in mapping strings

As noted in the beginning of this article, calculation operands can be used with the sequence mapping items **SD** and **SX**, and the numeric conversion mapping items **DD**, **DX**, **XD** and **XX** in mapping strings.

For a complete description of mapping strings, see [Working with Mapping Strings](#). For concise information on specific mapping commands, see [Mapping Strings Quick Reference Syntax and Examples](#).

Here are some brief examples.

1. Suppose you want to find some “words” and append a dash and a decimal numeric suffix to them. The first “word” gets a sequence number 1, the second get 2, etc. Let s say your data looks like this:

```
000151 ***** one    +++
000152 ***** two    +++
000153 ***** three   +++
000154 ***** four   +++

```

You can find “word” strings of varying length with a Regular Expression of **R' [a-z]+'**. In order for the *sequence number* mechanism of mapping strings to function, you will have to use **ALL** in your **CHANGE** command.

For the **M** string portion, you will want to do the following:

- emit the original word; you can do that with a **1+** mapping code
- emit a dash; this is done with a **' - '** literal
- create a sequence number value using the mapping command **SD**

If you want the shortest possible number used as the numeric suffix, you would use the Varying length format, with **SDV** or just **SD** alone. If you wanted a fixed number of digits (say, **4**) you would specify a command like **SD4F**.

Put that all together, and your **CHANGE** command would look like this:

```
CHANGE ALL R' [a-z]+' M"1+ '-' SD4F"
```

That would change your data as follows:

```
000151 ***** one-0001    +++
000152 ***** two-0002    +++
000153 ***** three-0003   +++
000154 ***** four-0004    +++

```

You will note that you can do all the above without the need of a *calculation operand*. But, suppose you didn t want the suffix to be a value from **1** by **1**, but a value from **0** by **20** plus a constant of **5**. The **SD** command alone can t do that, but used with a calculation operand, it can.

Recall from the description of the pre-initialized values for calculation variables, variable **S** will contain the value of the original *sequence number*. In this example, that original *sequence number* ranges from 1 to 4. To get the numbers the way you want, you would have to subtract one from the *sequence number*, multiply it by 20, and then add 5. To make that the *returned value*, you would then assign the result to variable **R**. You would specify this in a *calculation operand* as **R=(S-1)*20+5**. You incorporate the *calculation operand* into the mapping string and issue your command like this:

```
CHANGE ALL R' [a-z]+' M"1+ '-' SD4F'R=(S-1)*20+5'"
```

That would change your data as follows:

```
000151 ***** one-0005    +++
000152 ***** two-0025    +++
000153 ***** three-0045   +++
000154 ***** four-0065    +++

```

2. Continuing with the first example, suppose what you wanted was, not to append a sequential number, but the actual line number where the data lines are found. Let's say our data is similar to the first example, but there are excluded lines between them:

```
***** ***** Top of Data *****
----- < 000150 > -----
000151 ***** one    +++
----- < 000010 > -----
000162 ***** two    +++
----- < 000012 > -----
000175 ***** three   +++
----- < 000014 > -----
000190 ***** four   +++
***** ***** Bottom of Data *****

```

We can accomplish this basically the same way as in example 1. We will again use the **SD** command.

You might wonder how we can use **SD** if we are not going to use the sequence number in the value. The answer is that when **SD** has a *calculation operand*, the *calculation function* is called and variables get pre-initialized as described above. However, there is no requirement that **SD** or any of the other mapping commands that can use a *calculation operand* must **reference** variable **S** or any other variable. For our second example here, we will simply ignore variable **S**. The reason **SD** is the right command to use is that we want to "generate" a number that is **not** based on our input data – which here consists of *words*, rather than *numbers*.

The *calculation operand* here will be very simple. Variable **L** contains the line number where the **CHANGE** command finds the words in question. We just have to make sure than only non-excluded lines are affected, and change the *calculation operand* to copy the value of variable **L** to variable **R**.

You would issue your command like this:

```
CHANGE ALL NX R'[a-z]+' M"1+ '-' SD4F'R=L'"
```

That would change your data as follows:

```
***** ***** Top of Data *****
----- < 000150 > -----
000151 ***** one-0151    +++
----- < 000010 > -----
000162 ***** two-0162    +++
----- < 000012 > -----
000175 ***** three-0175   +++
----- < 000014 > -----
000190 ***** four-0190    +++
***** ***** Bottom of Data *****

```

3. Converting data to a hash key. This is just a made-up example, but it may give you some idea of what you can

do with calculation operands, when you have an elaborate computation to perform.

Suppose you had a series of 5-digit numbers, and you want to calculate a *hash value*. You have various issues to contend with:

- To make the process flexible, numbers > 99999 will be limited to 5 digits using an initial modulus, just to be safe. That is done with **x \ 1L5**.
- You calculate a second modulus into 100003, but some of your numbers could be zero, and you can't divide by zero. Instead, when the divisor is zero you force it to one.
- After taking a final modulus of 9973, the value will be in the range of 0 to 9972, but when the value was zero you want to force it to 9973.

Since we want to process original numeric data, this requires a **DD** command instead of **SD**. To keep the example simple, the data will be in fixed columns:

```
***** ***** Top of Data *****
000001 19453
000002 48470
000003 03085
000004 66247
***** ***** Bottom of Data *****
```

The numbers themselves are found with a picture of **P'#####' 1 5**, and variable **x** is set to this value. The result string will be 4 digits, so the basic **DD** command will appear as **DD4F**.

- The first modulus of **X** is calculated as **A=X\1L5**, to limit the values to 5 digits.
- Divide this into 100003, but if the value from the first step was zero, substitute 1 instead. Adding the conditional expression **A=0** to **A**, whenever **A** is zero the value would be replaced by 1. Tests for equality have lower precedence than addition, so this part must be in parentheses. This gives **B=100003\ (A+(A=0))**.
- The final modulus is calculated as **R=B\9973**.
- Forcing a zero value to **9973** requires a test for zero. Since assignments are also expressions that have the value assigned to the variable, we can take that assignment and compare it to zero. We have to use an explicit assignment of **:=** so the expression is understood not to be a comparison operation. If it is zero, we can use the result of the comparison to select a substitute value for **R** using a set expression that contains an assignment to **R**. When **R** is not zero, we simply select a dummy set clause of **0**, which is discarded, leaving the value of **R** unchanged.

This gives us the following calculation:

```
A=X\1L5 ; B=100003\ (A+(A=0)) ; ((R:=B\9973)=0) [R=9973,0]
```

The complete CHANGE command would be:

```
C P'#####' 1 5 M"DD4F'A=X\1L5 ; B=100003\ (A+(A=0)) ; ((R:=B\9973)=0)
[R=9973,0]"
```

Of course, a command this long is something you would prepare ahead of time. You can place the mapping string into an SPFLite SET variable, like this:

```
SET MYMAP = `M"DD4F'A=X\1L5 ; B=100003\ (A+(A=0)) ; ((R:=B\9973)=0)
[R=9973,0]"`
```

Then, issue the command as:

```
C P'#####' 1 5 =MYMAP
```

You can also compose long mapping strings in an SPFLite edit macro, and issue them using the **SPF_Cmd()** built-in function. The results of the CHANGE command would appear as follows. Note that these results were tested using the actual calculation function, and verified with an Excel spreadsheet:

```
***** ***** Top of Data *****  
000001 2738  
000002 3063  
000003 1283  
000004 3837  
***** ***** Bottom of Data *****
```

Calculator command-line test program

Because the use of a calculator in a text editor is a somewhat novel feature, you may need a little practice working with it to be sure you understand how it works. One way to do that is by creating experimental mapping strings with calculation operands, and observing how they operate on test data. However, that process could be somewhat time-consuming.

An easier way to explore this is with a small command-line test program called **mapping_calc.exe**.

This program will first prompt you for an initial sequence number (seq ->). If you use expressions that reference variable **s**, this will provide an incrementing value for each calc expression you supply, starting with the number you enter. If you just press Enter, the starting number will be **1**. If you provide a starting number of **0**, variable **s** will be set to **0** each time the *calculation function* is called, and the value will not be incremented.

You cannot directly set the values of the Line-number variable **L** or the Column-number variable **C**. Instead, **mapping_calc.exe** will assign random numbers to these values for testing purposes, with variable **L** receiving random numbers between 1 and 1000, and variable **C** receiving random numbers between 1 and 80.

The program will then prompt you for an initial value (init ->) and a calculation operand (calc ->). The initial value is used to initialize variables **R** and **x**.

You type these in as you would in a mapping string (just the calculation part). It will then return the calculated result in decimal (dec ->) and hex (hex ->), and report any errors it may have found.

If a specific column is associated with an error, it will appear at the beginning of the error message in parentheses. For example, if you specified a calculation operand of **R=x/\2** this would be an error, since both **/** and **** are binary operators, and there is no such operator as **/**. Since the error would be detected when the **** was reached in position 5 of the expression, you would see the error message (5): Calc operand has syntax error at ****.

To terminate the program, simply press Enter twice at the (init ->) prompt, or click on the [x] Windows button. In this program, hex values are entered and displayed with a leading **.** dot notation. You can use this hex format for both the initial value and in the calculation expression itself, as described above in the section Specifying numeric literals.

The **mapping_calc.exe** program is distributed with the SPFLite installation.

Mapping String Quick Reference Guide

Command Index

AQ	Insert Accent Quote into result string
AX	Perform Ansi to Hex character conversion
C	Center text in result string
CC	Set CASE C as default for character searches
CT	Set CASE T as default for character searches
D	Delete character columns from result string
DC	Delete character values from result string
DD	Perform Decimal to Decimal numeric conversion
DQ	Insert Double Quote into result string
DX	Perform Decimal to Hex numeric conversion
DV	Specify a Dynamic Variable as a mapping string item
EX	Perform EBCDIC to Hex character conversion
IC	Invert Case of result string
L	Left justify the result string
LC	Set result string to Lower Case
M	Move characters within result
P	Pad the result string on both sides
PL	Pad the result string on Left side
PR	Pad the result string on Right side
R	Right justify the result string
RA	Replace All
RC	Replace Characters via translation table
RL	Replace Left-most
RP	Repeat the result string
RR	Replace Right-most
RV	Reverse the contents of the result string
SC	Set result string to Sentence Case
S*	Generate Sequence number in Decimal and Hex – SD / SX
SQ	Insert Single Quote into result string
SV	Specify a Set Variable as an external mapping string item
TC	Set result string to Title Case
T	Trim the result string on both sides
TL	Trim the result string on Left side
TR	Trim the result string on Right side
UC	Set result string to Upper Case
X	Exchange strings
XA	Perform Hex to Ansi character conversion
XD	Perform Hex to Decimal numeric conversion
XE	Perform Hex to EBCDIC character conversion
XX	Perform Hex to Hex numeric conversion
Z	Zero-suppress the result field

General information

In syntax descriptions,

- *Italics* describe values you supply, or are references to other syntax
- **Bold** text describes parts of a mapping string you literally specify
- `[]` brackets enclose optional items; the brackets are not literally specified
- `{}` braces group items together; the braces are not literally specified
- `|` or-sign separates alternative; the or-sign is not literally specified
- `-` dashes in syntax names, like *command-code*, are for readability
- Dashes are not literally specified unless shown as – in bold face Courier font

In a mapping string, individual *items* are separated from each other by one or more blanks. Within a command-code *item* that has operands, the command code and its associated operands must have no embedded spaces or other values not part of the syntax, unless these are part of a quoted string operand.

A *mapping command* is a command used to copy and transform character data found by an SPFLite **CHANGE** Primary command. These mapping commands are specified in a **CHANGE** *string-2* operand string with a string type code of **M**.

The string value found by *string-1* of the **CHANGE** Primary command is called the *source string*. This value is referenced by some of the mapping commands. It is a read-only value that can be referenced or copied (in whole or in part), but cannot be modified.

The string produced as a result of processing the mapping command is called the *result string*. The *result string* initially is an empty (zero-length) string that is successively built up by process of executing *mapping commands*.

Mapping commands can be generally divided into two types: **copying commands** and **modifying commands**.

- A **copying command** copies characters from the *source string* to the *result string*, by appending one or more characters from the *source string* to the right-hand side of the *result string*. Sometimes a copying command will copy part of the *source string* and apply certain types of processing to it, so that the value placed in the *result string* is not always an exact copy, but may be transformed in some way.
- A **modifying command** alters the contents of the *result string* without reference to the *source string*.
- Some *modifying commands* produce string values copied from some part of the current *result string* and are appended to the end of it, making the *result string* longer. Other *modifying commands* modify the *result string* in place, but do not change its length. The Delete command **D** will make the *result string* shorter, and the Move command **M** could make the length longer, shorter or unchanged, depending how the value is moved.

When a *column reference* appears by itself without an associated *command code*, the *column reference* is considered a *copying command* that refers to data in the *source string*; it requires *dot notation* to make it refer to the current *result string*.

The distinction between *copying commands* and *modifying commands* affects how *column references* with *dot notation* are treated.

See [Full Mapping String Documentation](#) for a complete discussion of mapping strings.

Sample Usage with the CHANGE command

```
CHANGE  R" [A-Za-z] +"  M" 4-1 "  WORD  ALL
```

Mappings strings use an **M** type code, and always appear as *string-2* of a **CHANGE** command. The type code **M** may appear at beginning or end of *string-2*, and is case-insensitive. *String-1* will normally be a Picture (**P**) or Regular Expression (**R**) string type.

Character enumeration guide

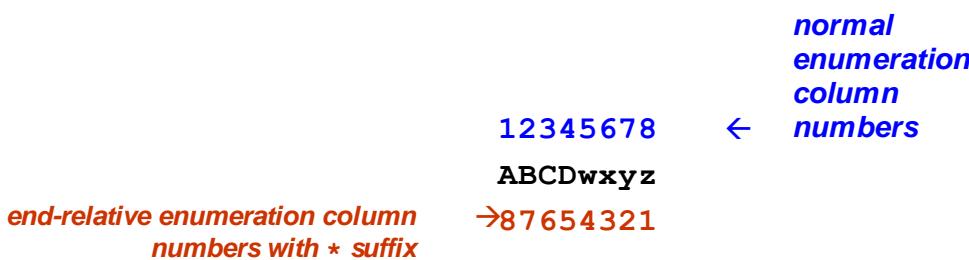
General

Column Reference Syntax

Dot Notation

General

Consider an example source string value of **ABCDwxyz**. The columns for this value are enumerated as follows:



Notes:

- Throughout this document, when column references appear in explanations and examples, they are **color-coded** to help remind you which type of column reference or enumeration type is being used.

Since there are a limited number of useful colors available, a given color may be used for more than one purpose; **red** will often but not always mean **end-relative**. Read the context to confirm the meaning.

- For fixed-length data, **normal enumeration** is the easiest to understand and use. Follow the examples below in **blue** for **normal column range notation**.
- When your goal is to access the right-hand “tail” portion of a value, the **remaining-columns notation** can be useful. To select an entire string value (like **ABCDwxyz** above) you can use a using **remaining-columns** reference with a **1+** notation. This will work for a string of any length.

When starting column number is 1, the **1 can be omitted** and you can **use the + plus by itself** to mean the same as **1+**. When **+** is used in this way, it is comparable to the **! Picture** code that copies the entire found string. A **+** plus sign, whether preceded by a number or alone, must always be followed by a blank, unless it's the last thing in your mapping string.

- When you are dealing with variable-length data, perhaps as found by a Regular Expression **find string** on your **CHANGE** command, the **end-relative** and **mixed-mode end-relative** notations can help you deal with data when you don't know its exact size in advance.
- When it is easier to view your data from the standpoint of its **right-hand end** (wherever that may be), **end-relative enumeration** is a convenient notation, in which your column numbers are “reversed” (as compared to how **normal** column notation does it), and an “*****” asterisk suffix is used. (By “reversed”, refer to the **red column numbers** in the diagram above.) Follow the examples below in **red** for **end-relative column range**

notation, noting carefully the column-number ordering that is used.

- For **mixed-mode end-relative notation**, the first (**left-hand**) number always uses **normal enumeration**, while the second (**right-hand**) number always uses **end-relative enumeration**. The separator for these numbers is a / slash to indicate forward column ordering, and is a \ backslash to indicate reverse column ordering. **Mixed-mode end-relative notation** can be useful in cases where, for example, you want to reference **all but** the first **2** and last **2** positions of a value and start with relative position **3** in both directions; you would do that with a **3/3*** column reference. Follow the examples below in **violet** for **mixed-mode end-relative column range notation**, observing carefully the special syntax it uses.

Note that **only** by using a \ backslash do you indicate column reversal in **mixed-mode enumeration**; you do **not** reverse the numbers to indicate column reversal, as is done with **normal enumeration**.

Examples using the source string **ABCDwxyz**:

To explicitly reference the **entire** source string normally as **ABCDwxyz**:

- 1-8** **normal column range notation**
- 8-1*** **end-relative column range notation**
- 1/1*** **mixed-mode end-relative column range notation, forward**
- 1+** **remaining-columns notation**

To explicitly reference the **entire** source string in reverse to obtain **zyxwDCBA**:

- 8-1** **normal column range notation**
- 1-8*** **end-relative column range notation**
- 1\1*** **mixed-mode end-relative column range notation, reversed order**
- \1+** **reversed order remaining-columns notation**

To locate substring **ABCD** within the source string:

- 1-4** **normal column range notation**
- 8-5*** **end-relative column range notation**
- 1/5*** **mixed-mode end-relative column range notation, forward**

To locate substring **wxyz** within the source string:

- 5-8** **normal column range notation**
- 4-1*** **end-relative column range notation**
- 5/1*** **mixed-mode end-relative column range notation, forward**
- 5+** **remaining-columns notation**

To locate and reverse substring **ABCD** within the source string to get **DCBA**:

- 4-1** **normal column range notation**
- 5-8*** **end-relative column range notation**
- 1\5*** **mixed-mode end-relative column range notation, reversed order**

To locate and reverse substring **wxyz** within the source string to get **zyxw**:

- 8-5** **normal column range notation**
- 1-4*** **end-relative column range notation**
- 5\1*** **mixed-mode end-relative column range notation, reversed order**

To locate substring **BCDwxy** within the *source string*:

- 2-7 **normal column range notation**
- 7-2* **end-relative column range notation**
- 2/2* **mixed-mode end-relative column range notation, forward**

To locate and reverse substring **BCDwxy** within the *source string* to get **yxwDCB**:

- 7-2 **normal column range notation**
- 2-7* **end-relative column range notation**
- 2\2* **mixed-mode end-relative column range notation, reversed order**

Column Reference Syntax

[*dot-notation*] *column-item*

Column-item: one of the following formats:

- 5 **single column, normal notation**
- 5* **single column, end-relative notation**
- 1-4 **column range, forward order, normal notation**
- 4-1 **column range, reversed order, normal notation**
- 8-5* **column range, forward order, end-relative notation**
- 5-8* **column range, reversed order, end-relative notation**
- 2/3* **column range, forward order, mixed-mode end-relative notation**
- 2\3* **column range, reversed order, mixed-mode end-relative notation**
- 5+ **remaining columns (like 5 and above), forward order**
- \5+ **remaining columns (like 5 and above), reversed order**
- 0 **as-yet-unreferenced source-string columns, forward order**
- \0 **as-yet-unreferenced source-string columns, reversed order**

When a *column reference* appears by itself without an associated *command code*, the *column reference* is considered a *copying command* that refers to data in the *source string*; it would require *dot notation* to make it refer to the current *result string*.

For example, to copy **column 1 to 4** of the *source string* into the *result string*, simply specify 1-4 without a command code. To copy **column 1 to 4** from the beginning of the *result string* and appending it to the end of that same *result string*, specify .1-4 without a command code.

When a *column reference* appears as part of a command code, it is treated like an *operand* to that command. See [Specific Command Coding Notes](#) for complete information on how each command code operates, and how it treats its operands.

Dot Notation

When a *column reference* is preceded by a **dot**, the reference is to the current contents of the *result string* at the time the reference is made.

Without the dot, it may reference to the original *source string* or to the *result string*, depending on whether a **copying command** or a **modifying command** is used.

- When a command is a copying command, *dot notation* causes a *column reference* to refer to the current contents of the *result string instead of the source string*.
- When a command is a modifying command, **all column references** refer to the current contents of the *result string regardless* of whether . dot notation is used or not. A *modifying command* always refers to the *result string*, and so “dot mode” is implied for such commands.

See description of each command code for the meaning of the *column reference* operand for that command.

1-4 may refer to **columns 1-4** in the *source string* **or** in the *result string*
.1-4 refers to **columns 1-4** in the *result string* **only**

Dot-notation cannot be combined with *unreferenced columns* (column number of **0**).

Auto-Reference and Auto-Copy features

When a *modifying command* is issued without a *column reference* and the current state of the *result string* is empty, the *modifying command* will automatically copy the full contents of the *source string*. This feature is called *auto-copy*.

When a *copying command* is issued without a *column reference* and the current state of the *result string* is empty, the *copying command* will automatically refer to the full contents of the *source string*. This feature is called *auto-reference*. Since the *copying command* might alter the contexts from the *source string* **before** placing it the *result string* (like the **AX** command does), an *auto-reference* sometimes, but not always, results in an *auto-copy* operation as well. Consult the description of each command to determine exactly how this is handled in each case.

In the descriptions of each command, a notation is made in the “Command type” entry as to whether the command will Auto Copy or Auto Reference the *source string* when no explicit *column reference* is specified.

When a command supports Auto Copy or Auto Reference, an implied column reference of **1+** is used. If that meets your needs, you do not need to specify **1+** explicitly. The Auto Copy and Auto Reference features allow you to simplify the coding of your mapping string when you are have just one command code in it.

When the current *result string* is not empty, the Auto Copy feature is not performed. That is because it would be unusual to completely copy the *source string* (perhaps for a second time) when something is already in the *result string*. If that is what you wish to do, you will have to actually specify **1+** as needed. Thus, **an implicit Auto Copy feature will only be applied for any given mapping string when the result string is empty, which normally will only occur once**. In contrast, the Auto Reference feature can be used as many times as needed in your mapping string.

Inserted text

Inserted text to be appended as a literal value at the end of the *result string*, or are a parameter of a command code, are specified as inner quoted strings within the larger M string. Example: `M"1-2 `**` 3-4"`.

Inserted text can have a replication factor; `M" `Abc` 2"` produces `'AbcAbc'`.

The type of quote used for such inner quoted values must differ from the outer quotes used on the M string itself; (') single, (") double and (`) accent quotes are available. To represent a quoted-quote as data in such inner quoted values, it must be doubled. Due to the way SPFLite handles its own quoted strings, you cannot use the quotes of the M string itself literally as data, neither by escaping them nor by doubling them. In the example above, the M string is enclosed in (") double quotes, so double quotes cannot be used as data anywhere within that M string. You can use the commands **SQ**, **DQ** and **AQ** to insert a Single Quote, Double Quote or Accent Quote. You can also insert text as a hex value, such as '31'X'.

Hex values can also have a replication factor; `M" `31`X4"` produces `'1111'`.

When a string parameter is involved in character comparisons, it can be suffixed with **C** or **T** to cause **CASE C** or **CASE T** comparisons to be performed. Example: `M"1+ RC`abc=xyz`T"`. Commands such as **RC** here that use a *string-pair* syntax can also perform *case-conformant* changes, depending on if and where the string type of **T** is used.

Hex strings can be specified with an X suffix, and are always treated as case-sensitive. String suffixes take precedence over **CC** or **CT** command codes, which take precedence over the Edit **PROFILE CASE** setting.

Inserted text ranges use a range of Ansi characters in brackets. Example: `[0-9]` produces the same as `'0123456789'` and `[9-0]` produces the same as `'9876543210'`. Multiple ranges are allowed. Example: `[A-Z,0-9]` produces the same as `'ABCDEFGHIJKLM NOPQRSTUVWXYZ0123456789'`.

Text Case Handling

Case Modification

Characters are copied from the *source string* to the *result string* as-is, unless you provide codes to modify the case of the data:

- > Case modification code causes all data placed in the *result string* from that point forward to be converted to upper case
- < Case modification code causes all data placed in the *result string* from that point forward to be converted to lower case
- >< Case modification code causes all data placed in the *result string* from that point forward have its case *inverted* such that ABCDwxyz will become abcdWXYZ. There can be no spaces between the > and < symbols.
- <> Case modification code causes all data placed in the *result string* from that point forward to resume being copied as-is without case modification. There can be no spaces between the < and > symbols.
- When mapping commands are specified while a > , < or >< case modification code is in effect, certain commands may handle these codes differently or may disregard them. Consult specific commands for more information.
- **UC, LC, SC, TC** and **IC** case alteration command codes alter the current contents of the *result string* at the point the command is encountered, performing Upper Case, Lower Case, Sentence Case, Title Case and Invert Case operations, respectively, on the *result string*. Refer to the main Help guide for the Primary and Line commands of SC and TC to understand the capitalizations these perform.

Case Sensitivity and Case Conformance

When a string operand appears on a command code, and that command involves searching for or changing data, the command will apply comparison and change rules that are similar to how SPFLite in general handles this when it executes a **FIND** or **CHANGE** primary command.

Mapping commands handle case sensitivity as follows:

- If you have provided no other indications on your mapping string as to how you want string searches to be handled, the mapping-string facility will use the current **CASE** setting from the **PROFILE** setting of the file you were editing when you issued the **CHANGE** command that contained a mapping string. If your **PROFILE** setting is **CASE C**, string comparisons will be done as case-sensitive; if your **PROFILE** setting is **CASE T**, string comparisons will be done as case-insensitive.
- If you specify a mapping command code of **CC** or **CT**, it overrides the **PROFILE CASE** setting. If you specify a mapping command code of **CC**, string comparisons from that point forward will be done as case-sensitive, unless that is changed by a subsequent **CT** command. If you specify a mapping command code of **CT**, string comparisons from that point forward will be done as case-insensitive, unless that is changed by a subsequent **CC** command.

- If you specify a string operand on a command code, and that string contains a trailing type code of **C** or **X**, it implies a case-sensitive comparison, which overrides any **CT** mapping command or **PROFILE CASE T** setting which may be in effect. If you specify a string operand on a command code, and that string contains a trailing type code of **T**, it implies a case-insensitive comparison, which overrides any **CC** mapping command or **PROFILE CASE C** setting which may be in effect.

Some commands, such as **RA** (Replace All) accept a two-part string operand syntax, consisting of a *first* and a *second* string value. Such commands involve both *finding* and *changing* data.

When data is changed with such commands, it is possible to cause such changes to occur in a *case conformant* manner, the same as is done when a **CHANGE** primary command has string operands with string type codes of **T** for both string-1 and string-2. See Case-Conformant Change Strings under Finding and Changing Data in the Help topic Working with SPFLite for more information on how the *case conformant* change process operates.

For mapping commands that use **pairs** of strings, these special operands work as shown below, using the **RA** command as an example, which use a pair of *first* and *second* string values.

You will see that there are a large number of cases. Instead of trying to memorize these cases, it is best to observe that, with few exceptions, mapping commands handle strings the same way as **FIND** and **CHANGE** primary commands do when their *string-1* and *string-2* operands use the same type codes. Where there are **differences** from the way primary commands handle this, those differences are **highlighted in red**.

Note that after an **X** value is converted to its character equivalent, it is treated like type **C**; that is, as case-sensitive for comparisons, and an **X** value is used as-is when specified as *string-2*.

#	Sample Command	String-1 type	String-2 type	How <i>string-1</i> compared	How <i>string-2</i> changed
1	RA' <i>first=second</i> '	omitted	omitted	Based on CC/CT command in effect, or on current PROFILE CASE setting	As specified, as if <i>string-2</i> had a type code of C
2	RA' <i>first=second</i> C RA' <i>first=second</i> X	C implied X implied	C X	Case sensitive	As specified
3	RA' <i>first=second</i> T	T implied	T	Case insensitive	Case conformant
4	RA' <i>first='second</i> C RA' <i>first='second</i> X	omitted	C X	Based on CC/CT command in effect, or on current PROFILE CASE setting	As specified
5	RA' <i>first='second</i> T	T implied	T	Case insensitive	Case conformant. Type T is implied for <i>string-1</i> because without it, the case conformance of <i>string-2</i> would serve no purpose.
6	RA' <i>firstT='second</i> '	T	omitted	Case insensitive	As specified, as if <i>string-2</i> had a type code of C . Note that type T on <i>string-1</i> is not assumed for <i>string-2</i> .

					2
7	RA'first'T ='second'C RA'first'T ='second'X	T T	C X	Case insensitive	As specified
8	RA'first'C='second' RA'first'C ='second'C RA'first'C ='second'X RA'first'X='second' RA'first'X ='second'C RA'first'X ='second'X	C C C X X X	omitted C X omitted C X	Case sensitive	As specified
9	RA'first'C='second' RA'first'C ='second'C RA'first'C ='second'X	omitted	omitted C X	Based on CC/CT command in effect, or on current PROFILE CASE setting	As specified
10	RA'first'C ='second'T RA'first'X ='second'T	C X	T T	Case sensitive	Case conformance is required for SPFLite compliance. In ISPF, the type T would be ignored.
11	RA'first':second	T <i>implied.</i> Type T is always assumed .	T <i>implied.</i> Type T is always assumed	Case insensitive	Case conformant. When <u>colon notation</u> is used, type T is implied for both strings. If you combine this colon notation with non-type-T codes, their values are accepted as written (including X), but strings are still treated as case-insensitive and case-conformant.
12	RA'first:second RA'first:second'C RA'first:second'T RA'first:second'X This is probably an invalid string specification. The colon does not delimit two strings, but is part of the data in the <u>one</u> and <u>only</u> string that is defined.	omitted C T X	omitted	Based on CC/CT command in effect, or on current PROFILE CASE setting	The command does not define <u>string-2</u> , but only <u>string-1</u> . The : colon in <u>string-1</u> is not a delimiter but is <u>ordinary data</u> , and has no special meaning. For the RA command, this would be a string-delete request. In other commands, it may be

If used with type X, the hex data will be invalid.

a syntax error.

Command-code syntax

General command syntax:

column-reference

or

command-code [*string-operand*] [*column-reference* | *numeric-operand*]

Column-reference:

See paragraph 4 above for description. A column reference by itself (without a *command-code*) selects one or more columns from the *source string* and appends them to the end of the *result string*.

When a **.dot** precedes a column reference by itself, it selects one or more columns from the current value of the *result string* and appends them to the end of that *result string*. When a *column-reference* appears on a *modifying command*, it always refers to the *result string*, whether a dot is used or not; the dot is implied in such cases.

A column reference following a command code usually selects a set of columns of the *source string* to be acted on by the command. See the description of the specific command for details on what the column reference is used for.

Command-code:

One or more letters like L or TR, defining a mapping action to be performed. Command codes for **DD**, **DX**, **XD** and **XX** allow an extended syntax; see paragraph 11 below.

String-operand:

An optional quoted string value, used by a command code. This will generally have the same syntax as *inserted-text* does; see paragraph 7 above for more information. See the description of the specific command for details on what this string operand may be used for. Some commands allow the *string operand* to be specified as a pair; see paragraph 9 above for more information. Certain commands will accept a **string set** or **string pair** as a *string operand*.

Numeric-operand:

An optional decimal value, used by some command codes, **instead of a column reference**. For example, the Pad command code **P** uses the *numeric operand* to specify a single decimal value as the number of padding characters to be used.

For some commands, the *string-operand* and/or the *column-reference* may be either mandatory or disallowed. See individual command descriptions for more information.

Extended command-code syntax for DD, DX, XD and XX

The command codes **DD**, **DX**, **XD** and **XX** can be used in a “standard” syntax or in an “extended” syntax. In standard syntax, the *column reference* operand selects the entire numeric field for numeric conversion and reformatting.

In extended syntax, the selected columns are scanned (**left-to-right** or **right-to-left**) for numeric digits delimited by non-digit characters. In a given string that is selected, up to 9 numeric fields within it can be scanned for, using an *ordinal number* (the **n** value below) of 1 through

The specific placement of the **n** value within the command code determines whether the scan goes **left-to-right**,

or **right-to-left**. When the **n** value is **left** of the first character of the command code (like **2DX**), the scan goes **left-to-right**. When the **n** value is **right** of the first character of the command code (like **D2X**), the scan goes **right-to-left**. See the examples below. (If the **n** value is specified as **0** it will be ignored.)

Once a value is found, you determine the *width* of the converted value you want to allow for (the **w** value below) and the *type* of value you want to produce (the **t** value below).

- Type **F** is fixed width, with leading “0” characters on the left.
- Type **Z** is fixed width, with leading blank characters on the left (zero-suppressed).
- Type **V** is variable width; **n** here means the minimum width.

Using **DX** as an example, this command can appear as **nDXwt** or **DnXwt**

- **3DX2F** scans for the **third** decimal number going **left-to-right**, converts it to hex, and produces a Fixed length field of 2 positions, which is zero-filled on the left.
- **D3X4Z** scans for the **third** decimal number going **right-to-left**, converts it to hex, and produces a Fixed length field of 4 positions, which is space-filled on the left.
- **1DXV** scans for the **first** decimal number going **left-to-right**, converts it to hex, and produces a Variable-length field at least one character long.

General examples

- (1) Have strings like ABCD and want DCBA. Assume words are case-insensitive.

CHANGE P'@@@@' WORD M'4-1'

- (1.1) The same example, using a RegEx with "quantifier" notation for the number of letters:

CHANGE R' [A-Z]{4}' WORD M'4-1'

- (2) Have strings like ABCD and want DCBA. Assume words are upper-case only.

CHANGE P'>>>' WORD M'4-1'

- (3) Have strings like ABCD and want ** AB-CD **

CHANGE P'>>>' WORD M'`` 1-2 ` - ` 3-4 ` **` '**

or

CHANGE P'>>>' WORD M'1-2 ` - ` 3-4 P P`*`2'

- (4) Have AB1234 and want XY4321

CHANGE P'AB####' M" `XY` 6-3"

- (5) Have XX1234 and want XX4321 where XX are any existing characters

CHANGE P'==####' M"1-2 6-3"

- (6) Have XX1234 and want xx4321 where XX, xx are upper and lower case

CHANGE P'>>####' M"< 1-2 >> 6-3"

- (7) Have AB1234 and want A123

CHANGE P'AB####' M"1 3-5"

- (8) Have AB1234 and want 123ABC

CHANGE P'AB####' M"3-5 1-2 `C`"

- (9) Have AB1234 and want BB234

CHANGE P'AB####' M"2 2 4-6"

- (10) Have strings like AB*12/34 and want AB1234*/ where * and / are some delimiters; we use 0 to "grab" **all the column positions not otherwise specified** and put them at the end, which are columns **3** and **6** in this example:

CHANGE P'@#=##=##' M"1-2 4-5 7-8 0"

- (11) Have 'words' of varying sizes; want to reverse first 3 characters and make upper case, then copy the rest of the string as-is. Example: abc12XyZ → CBA12XyZ

```
CHANGE R' [a-zA-Z]+ WORD M"> 3-1 <> 4+"
```

(12) Have ‘words of varying sizes; want to reverse first 3 characters and make upper case, then copy the rest of the string in reverse order in lower case. Example: abc12xyz → CBAzyx21

```
CHANGE R' [a-zA-Z]+ WORD M"> 3-1 < \4+"
```

Example of mapping ‘short data’

Have ‘words of varying sizes; want to reverse first 5 characters, and drop any characters after position 5.

```
CHANGE R' [a-zA-Z]+ WORD M"5-1"
```

What happens here if the word found is shorter than **5** characters? Suppose the string found is ABC. For mapping purposes, the string is treated *in effect as if* it were extended to the right with “null” characters, like this:
ABC

Then, the string order is reversed, like this:

CBA

Finally, the “null” characters are removed, and the result is CBA.

Note: The explanation above is just to help you understand. In reality, no “null” characters are actually used, so you can use mapping strings to reorder any data, even data having binary zeros. The character reversal process just goes from position **5** to position **1** as requested, but when positions **5** and **4** are asked for, it is found that there is no data in those positions and the copying action is simply ignored. So, internally, the command gets treated *for this particular string as if* you had specified this:

```
CHANGE R' [a-zA-Z]+ WORD M"3-1"
```

Examples of DX and XD numeric conversion commands

Suppose you have formatted strings of the form **R=nn,nn,nn** where **nn** are spans of decimal digits, but the precise number of digits can vary. The strings are to be converted into the form **Bxhhhhhh**, where each **h** is a hex digit. (You may recognize this as the RGB and BGR color palette codes from SPFLite's automatic colorization files.)

First, we need to find spans of characters that look like the ones we are looking for. There are a number of ways this can be done, but the simplest is to look for the **R=** prefix, followed by one or more commas and digits. This is not a precise definition, but it should be close enough. Our Regular Expression *find string* would thus look like **R'R=[0-9,]+'**.

- Note: Letters in a Regular Expression are matched to data as case-insensitive by default, unless your **R** string begins with a **\c** escape code. In our example, the data itself is case-insensitive, so we are fine with allowing this assumption to be used.
- Note: If you are unsure how “good” your Regular Expression string actually is, the best way to find out is to exclude your whole file, and then use the Regular Expression string in a **FIND ALL** command. If you like what “pops out”, you can continue with using it in a **CHANGE**; otherwise you may have to fix your Regular Expression definition.

The *result string* requires a constant string of **Bx** to be included in the result. We do that with an inner quoted string of **`Bx`**.

Next, we have to extract the decimal values, one at a time, and emit them as hex numbers with a fixed **format length** of **2** each. Our scan will proceed left-to-right, and we will be extracting the first, second and third decimal values found in that scan. So, each time is of the form **nDX2F1+**. These items have the following meaning:

- **n** is the relative location of the desired field. Because **n** appears on the **left side** of the command code, a **left-to-right** scan will be performed. There will be three mapping items, and because the *result string* has to have these values reversed, the **n** value used will be **3, 2** and **1** in that order.
- **DX** is the mapping conversion code, requesting Decimal to Hex conversion.
- **2F** requests a converted value being stored in a fixed field of two hex digits. By default, **DX** will render hex digits larger than **9** as capital **A-F**. You can use a case conversion code of **<** or one of the case alteration codes like **LC** to change that assumption. Our example will accept the default.
- The column reference operand **1+** selects the entire value found by the Regular Expression of **R'R=[0-9,]+'**. As you will see, this same value is referenced three times, since three separate scans are required to locate the first, second and third values from the original *source string*. Because the numbers could be anywhere in the *source string*, the entire value has to be scanned three times.

Combining this all together gives us the following **CHANGE** command:

```
CHANGE R'R=[0-9,]+' WORD M" `Bx` 3DX2F1+ 2DX2F1+ 1DX2F1+"
```

Because the **DX** command is a *copying command* that will *auto-reference*, the **1+** reference can be omitted, which simplifies the **CHANGE** command to:

```
CHANGE R'R=[0-9,]+' WORD M" `Bx` 3DX2F 2DX2F 1DX2F"
```

Example: A string **R=153,85,15** will be converted into **Bx0F5599**. The order of the numeric values has been

reversed, the values were converted from decimal to hex, each with a **fixed field size of 2**, the prefix `Bx` has been inserted, and the remainder of the original source data (the `R=` part) has been ignored. The entire found field was scanned three times to locate the values, because of the `1+` column reference that was used (or implied) three times.

Since the `n` values (3, 2, 1) precede the `DX` command codes (`3DX`, `2DX`, `1DX`), the scan is performed **left-to-right** (finding the third value from the left, second value from the left, then first value from the left).

If these had been written as (`D1X`, `D2X`, `D3X`) the **scan would have been performed right-to-left**.

Extra credit: Did you notice that we made no attempt to bypass the '`R=`' part of the *source string*? Why does this work? When your *source string* value is being scanned for numbers, characters like '`R`', '`=`' and '`'`' are not recognized as digits, so they are treated as separators between spans of digits, and otherwise are simply ignored.

Command Specific Syntax and Examples

Note: In order to be brief, these command summaries do not explain every detail. For more information, consult the main article, [Full Mapping String Documentation](#).

Command Index

AQ	Insert Accent Quote into result string
AX	Perform Ansi to Hex character conversion
C	Center text in result string
CC	Set CASE C as default for character searches
CT	Set CASE T as default for character searches
D	Delete character columns from result string
DC	Delete character values from result string
DD	Perform Decimal to Decimal numeric conversion
DQ	Insert Double Quote into result string
DX	Perform Decimal to Hex numeric conversion
DV	Specify a Dynamic Variable as a mapping string item
EX	Perform EBCDIC to Hex character conversion
IC	Invert Case of result string
L	Left justify the result string
LC	Set result string to Lower Case
M	Move characters within result
P	Pad the result string on both sides
PL	Pad the result string on Left side
PR	Pad the result string on Right side
R	Right justify the result string
RA	Replace All
RC	Replace Characters via translation table
RL	Replace Left-most
RP	Repeat the result string
RR	Replace Right-most
RV	Reverse the contents of the result string
SC	Set result string to Sentence Case
S*	Generate Sequence number in Decimal and Hex – SD / SX
SQ	Insert Single Quote into result string
SV	Specify a Set Variable as an external mapping string item
TC	Set result string to Title Case
T	Trim the result string on both sides
TL	Trim the result string on Left side
TR	Trim the result string on Right side
UC	Set result string to Upper Case
X	Exchange strings
XA	Perform Hex to Ansi character conversion
XD	Perform Hex to Decimal numeric conversion
XE	Perform Hex to EBCDIC character conversion
XX	Perform Hex to Hex numeric conversion
Z	Zero-suppress the result field

AQ - Insert Accent Quote into result string

Summary	Insert Accent Quote into result string
Command type	Modifying command; will not Auto-Copy
Case mod codes	Not applicable
Syntax	AQ [<i>replication-factor</i>]
<i>replication-factor</i>	Decimal value indicating the number of accent quotes inserted. If omitted, a one quote is inserted. If specified as zero, no quotes are inserted.
Example command	C P"@@@@" M`AQ 1+ AQ`
Source string	ABCDE
Result string	`ABCDE`
Remarks	This command is useful to insert one or more accent quotes when the outer M string is itself enclosed in accent quotes. Since this command does not auto-copy, you must explicitly specify other values you want in the result string, as is done with the 1+ code in the example.

AX - Perform Ansi to Hex character conversion

Summary	Convert character data to its Ansi encoding as hex digits
Command type	Copying command; will Auto-Reference
Case mod codes	Codes > and < honored; codes <> and >< treated as >
Syntax	AX [<i>bypass-string</i>] <i>column-reference</i>
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
<i>bypass-string</i>	Characters in <i>bypass-string</i> are not converted, but are copied as-is to the result string
Example command	C P'##=@@' M"AX`-` 1+ "
Source string	12-Lb
Result string	3132-4C62
Remarks	Note ‘-’ was not converted but copied. “L” converts to “4C” instead of “4c”. Default for hex digits > 9 is upper case.

C - Center text in result string

Summary	Current contents of <i>result string</i> are centered within a specified size
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	C [<i>pad-character</i>] <i>field-size</i>
<i>pad-character</i>	1-character value used for padding; if omitted, blank padding done
<i>field-size</i>	new length of <i>result field</i>
Example command	C P"@@@@" M" 1+ C8"

d	
Source string	ABCDE
Result string	° ABCDE ° °
Remarks	If <i>field-size</i> results in odd amount of padding, shorter amount goes on left. The ° symbol represents one blank. To pad with asterisk, use M"1+ C`*`8"

CC - Set CASE C as default for character searches

Summary	Sets case-sensitive CASE C mode for commands that do character comparisons
Command type	Case control; does not copy or modify <i>result string</i>
Case mod codes	Not applicable
Syntax	CC
Example command	C P'@@=@@=@@' M"CC 1+ RC`Aa=12`"
Source string	AA,aa,BB
Result string	11,22,BB
Remarks	To use CASE C mode on individual command codes, use C suffix on string: C P'@@=@@=@@' M"1+ RC`Aa=12`C" In absence of CC/CT or string suffix codes, PROFILE CASE option is used.

CT - Set CASE T as default for character searches

Summary	Sets case-insensitive CASE T mode for commands that do character comparisons
Command type	Case control; does not copy or modify <i>result string</i>
Case mod codes	Not applicable
Syntax	CT
Example command	C P'@@@@' M"CT 1+ RC`a=2`"
Source string	AA,aa,BB
Result string	22,22,BB
Remarks	To use CASE T mode on individual command codes, use T suffix on string: C P'@@@@' M"1+ RC`a=2`T" In absence of CC/CT or string suffix codes, PROFILE CASE option is used.

D - Delete character columns from result string

Summary	Delete specified character columns from current contents of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax column	D <i>column-reference</i>
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not the <i>column-reference</i> cannot be omitted
Example command	C P'@@@@' M"1+ D2-3"

Source string	AbcD
Result string	AD
Remarks	<p>D only applies to <i>result string</i>, not to <i>source string</i>. The <i>column-reference</i> is always treated as if it used <i>dot-notation</i> (implying use of <i>result string</i>) even if dot is omitted. Following command has same result: C P'@@@@' M"1+ D.2-3"</p> <p>Note: If you perform several D commands in the same mapping string, the relative column numbers to the right of where you delete data will <u>change</u> with each D command. To simplify this and avoid problems, do such multiple deletes in a right-to-left manner.</p>

DC - Delete character values from result string

Summary	Delete one or more instances of specified types of characters from <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	DC <i>character-deletion-list</i> [<i>column-reference</i>]
<i>character-deletion-list</i>	a quoted string of the form 'characters'
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> may potentially be modified
Example command	C P'=@@@@=' WORD M"1+ DC`)(`"
Source string	(AbCd)
Result string	AbCd
Remarks	The <i>characters</i> are one or more individual characters to be removed, and may be specified in any order.

DD - Perform Decimal to Decimal numeric conversion

Summary	Copy decimal data and apply numeric formatting
Command type	Copying command; will Auto-Reference
Case mod codes	Disregarded
Syntax	{ DD <i>left-to-right-cmd</i> <i>right-to-left-cmd</i> } [<i>format</i>] [<i>sign</i>] <i>column-reference</i>
<i>left-to-right-cmd</i>	nDD
<i>right-to-left-cmd</i>	DnD
n	A value from 1 to 9 , indicating the relative position within the selected columns where the number is found after scanning for it, where 1 means first, and 9 means ninth. When n is used as nDD , the scan for the numeric value is done in a left-to-right manner, and when n is used as DnD , the scan for the numeric value is done in a right-to-left manner.
	If n value is omitted, the DD command expects the full selected column range to be a single decimal number and no scan is performed to locate the value.
<i>format</i>	[w] t
	w - width of result number; w is required for format types F and Z , optional for type V If

	omitted for type v , w is assumed to be 1 . If omitted for types F and Z , w is taken as the same width of the original number as found in the <i>source string</i> .
t - format type code	format F is fixed-width with zero-fill or truncation on left format Z is fixed-width with zero-suppression or truncation on left format v is variable width with zero-fill on left; for v code, w means minimum width and a w value of 0 is allowed
<i>sign</i>	If sign code s is present, any leading + or – sign found during processing will be retained and copied to <i>result string</i> . If sign code s is absent, any leading + or – signs are ignored. Any spaces after the leading + or – sign are considered to be part of the sign.
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C P'###' WORD M"DD5Z1+"
Source string	123
Result string	°°123
Remarks	To produce 00123 use M"DD5F1+" The value 5 is the format length . The ° symbol represents one blank.

DQ - Insert Double Quote into result string

Summary	Insert Double Quote into result string
Command type	Modifying command; will not Auto-Copy
Case mod codes	Not applicable
Syntax	DQ [<i>replication-factor</i>]
<i>replication-factor</i>	Decimal value indicating the number of double quotes inserted. If omitted, a one quote is inserted. If specified as zero, no quotes are inserted.
Example command	C P"@@@@" M"DQ 1+ DQ"
Source string	ABCDE
Result string	"ABCDE"
Remarks	This command is useful to insert one or more double quotes when the outer M string is itself enclosed in double quotes. Since this command does not auto-copy, you must explicitly specify other values you want in the result string, as is done with the 1+ code in the example.

DV - Specify a Dynamic Variable as a mapping string item

Summary	The <i>result string</i> is used as a Dynamic Value of a mapping item
Command type	Modifying command; will Auto-Copy
Case mod codes	Disregarded
Syntax	DV [<i>delete-option</i>] <i>column-reference</i>
<i>delete-option</i>	An optional quoted string operand. If present, it contains a single letter specifying what is done with the <i>result string</i> prior to the DV command finishing.

	<p>The letter is case-insensitive.</p> <ul style="list-style-type: none"> ○ 'K' means to keep the <i>result string</i> as-is without change. ○ 'D' means to delete the portion of the <i>result string</i> that was referenced by the <i>column-reference</i> operand. This is the default if the <i>delete-option</i> is omitted. ○ 'E' means to erase the entire <i>result string</i> so that it contains a null (empty) string as it does when mapping string is first processed.
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is used as the source of the new mapping command
Remarks	See the main Help article, Working with Mapping Strings for more information on this command.

DX - Perform Decimal to Hex numeric conversion

Summary	Convert decimal data to hex and apply numeric formatting
Command type	Copying command; will Auto-Reference
Case mod codes	Codes > and < honored; codes <> and >< treated as >
Syntax	<code>{ DX left-to-right-cmd right-to-left-cmd } [format] [sign] column-reference</code>
<i>left-to-right-cmd</i>	nDX
<i>right-to-left-cmd</i>	DnX
n	A value from 1 to 9 , indicating the relative position within the selected columns where the number is found after scanning for it, where 1 means first, and 9 means ninth. When n is used as nDX , the scan for the numeric value is done in a left-to-right manner, and when n is used as DnX , the scan for the numeric value is done in a right-to-left manner.
<i>format</i>	If n value is omitted, the DX command expects the full selected column range to be a single decimal number and no scan is performed to locate the value. <code>[w] t</code> w - width of result number; w is required for format types F and Z , optional for type V If omitted for type V , w is assumed to be 1 . If omitted for types F and Z , w is taken as the same width of the original number as found in the <i>source string</i> . t - format type code <ul style="list-style-type: none"> • format F is fixed-width with zero-fill or truncation on left • format Z is fixed-width with zero-suppression or truncation on left • format V is variable width with zero-fill on left; for V code, w means minimum width and a w value of 0 is allowed
<i>sign</i>	If sign code S is present, any leading + or - sign found during processing will be retained and copied to <i>result string</i> . If sign code S is absent, any leading + or - signs are ignored. Any spaces after the leading + or - sign are considered to be part of the sign.
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C P'###' WORD M"DX2Z1+"

Source string	015
Result string	0F1
Remarks	To produce F use M"DX2F1+" The value 2 is the format length . See Working with Mapping Strings to Reorder, Reformat and Transform Data for a description of how the casing of hex digits > 9 is rendered.

EX - Perform EBCDIC to Hex character conversion

Summary	Convert character data to its EBCDIC encoding as hex digits
Command type	Copying command; will Auto-Reference
Case mod codes	Codes > and < honored; codes <> and >< treated as >
Syntax	EX [<i>bypass-string</i>] <i>column-reference</i>
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
<i>bypass-string</i>	Characters in <i>bypass-string</i> are not converted, but are copied as-is to the <i>result string</i>
Example command	C P'##=@@' M"EX`-`1+"
Source string	12-Lb
Result string	F1F2-D383
Remarks	Note ‘-’ was not converted but copied. “L” converts to “D3” instead of “d3”. Default for hex digits > 9 is upper case. The EBCDIC translation used will be the standard 1140 code page, without regard to the SOURCE code page setting.

IC - Invert Case of result string

Summary	Inverts the case of the <i>result string</i> , as of the point the IC command is issued
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	IC [<i>column-reference</i>]
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is set to lower case
Example command	C P'@@@@' M"1+ IC"
Source string	AAcc
Result string	aaCC
Remarks	

L - Left justify the result string

Summary	Left-justify contents of <i>result string</i> within a specified size
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	L [<i>pad-character</i>] <i>field-size</i>
<i>pad-character</i>	1-character value used for padding; if omitted, blank padding done

field-size	new length of <i>result field</i>
Example command	<code>C P"@@@@" M"1+ L8"</code>
Source string	ABCDE
Result string	ABCDE^ ^ ^
Remarks	The <code>^</code> symbol represents one blank. To pad with asterisk, use <code>M"1+ L`*`8"</code>

LC - Set result string to Lower Case

Summary	Sets contents of <i>result string</i> to lower case, as of the point the <code>LC</code> command is issued
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	<code>LC [column-reference]</code>
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is set to lower case
Example command	<code>C P'@@@@" M"1+ LC"</code>
Source string	AAcc
Result string	aacc
Remarks	

M - Move characters within result

Summary	Move data from one location to another within result string
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	<code>M destination-reference column-reference</code>
<i>destination-reference</i>	<ul style="list-style-type: none"> ○ when this is a single column number as <i>n</i>, it defines a column <i>n</i> within the <i>result string</i> where the data is to be moved into, with existing columns <i>n</i> and beyond being pushed to the right ○ when this is a single column number as <i>x-y</i>, it defines a column range within the <i>result string</i> where the data is to be moved into, with existing <i>x-y</i> columns being deleted ○ <i>destination-reference</i> may be <i>end relative</i>. If specified as a lone <code>*</code> asterisk, data is moved to the end of the result string.
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not the <i>column-reference</i> cannot be omitted
Example command	<code>C R' [-A-Fa-f0-9]+ WORD M"1+ M`5`9-11"</code>
Source string	One/\TwoXyz
Result string	One/Xyz\Two
Remarks	The Move command would be used when you need to rearrange the <i>result string</i> , or when the default of appending values to the end of the <i>result string</i> is not what you need. The destination string <code>5</code> means the moved data is inserted at position <code>5</code> of the current <i>result string</i> , after the point where the <code>\</code> is located, and the remaining data is pushed to the right.

	9-11 selects the "xyz" from the current <i>result string</i> . These columns are deleted after their value is moved to the specified location. Thus, if you move columns from the middle of the string to the end of it, you reference the column numbers as of prior to the move being performed.
--	---

P - Pad the result string on both sides

Summary	Add <i>pad characters</i> on both the left and right sides of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	P [<i>pad-character</i>] [<i>pad-amount</i>]
<i>pad-character</i>	If omitted, the pad character is a single blank
<i>pad-amount</i>	Number of pad characters to be added to each side. If omitted, 1 is assumed.
Example command	C P'@@@@' M"1+ P"
Source string	AAcc
Result string	°AAcc°
Remarks	The ° symbol represents one blank. To pad with asterisk, use M"1+ P`*`"

PL - Pad the result string on Left side

Summary	Add <i>pad characters</i> on left side of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	PL [<i>pad-character</i>] [<i>pad-amount</i>]
<i>pad-character</i>	If omitted, the pad character is a single blank
<i>pad-amount</i>	Number of pad characters to be added to each side. If omitted, 1 is assumed.
Example command	C P'@@@@' M"1+ PL"
Source string	AAcc
Result string	°AAcc
Remarks	The ° symbol represents one blank. To pad with asterisk, use M"1+ PL`*`"

PR - Pad the result string on Right side

Summary	Add <i>pad characters</i> on right side of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	PR [<i>pad-character</i>] [<i>pad-amount</i>]
<i>pad-character</i>	If omitted, the pad character is a single blank

<i>pad-amount</i>	Number of pad characters to be added to each side. If omitted, 1 is assumed.
Example command	<code>C P'@@@@' M"1+ PR"</code>
Source string	AAcc
Result string	AAcc°
Remarks	The ° symbol represents one blank. To pad with asterisk, use <code>M"1+ PR`*`"</code>

R - Right justify the result string

Summary	Right-justify contents of <i>result string</i> within a specified size
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	<code>R [pad-character] field-size</code>
<i>pad-character</i>	1-character value used for padding; if omitted, blank padding done
<i>field-size</i>	new length of <i>result field</i>
Example command	<code>C P"@@@@" M"1+ R8"</code>
Source string	ABCDE
Result string	°°°ABCDE
Remarks	The ° symbol represents one blank. To pad with asterisk, use <code>M"1+ R`*`8"</code>

RA - Replace All

Summary	Replace all instances of <i>first</i> string value with <i>second</i> string value
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy. Otherwise, codes > and < honored; codes <> and >< disregarded in favor of explicit string type codes.
Syntax	<code>RA string-values [column-reference]</code>
<i>String-values</i>	A quoted string of the form ' <i>first</i> = <i>second</i> ' and may contain a string type code. The <i>first</i> and <i>second</i> parts can be of different lengths; the <i>second</i> part can be null, but the <i>first</i> part cannot. When <i>second</i> string is null, command acts to delete the <i>first</i> string. The <i>before</i> and <i>after</i> parts cannot contain = equal signs as data if specified in the single <i>string-values</i> form.
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> may potentially be modified
Example command	<code>C P'=====' WORD M"RA`pay=taxes`"</code>
Source string	gross pay,net pay
Result string	gross taxes,net taxes
Remarks	When characters to the left of = sign contain letters, see main article for explanation how the case of letters is handled. The <i>first</i> and <i>second</i> parts can be specified as a <i>string pair</i> , example: <code>M"RA`pay`=`taxes`"</code> . When this is done, each part can have its own string type code of C, T or X, and can contain = equal signs as data. A string value

	with a <i>string-pair separator</i> of ":" instead of "=" signifies a <i>case-conformant</i> change. See section 9, Case sensitivity and case conformance , for an explanation of how type codes on string pairs are handled.
--	---

RC - Replace Characters via translation table

Summary	Modify <i>result string</i> by translating one set of characters with another
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy. Otherwise, codes > and < honored; codes <> and >< disregarded in favor of explicit string type codes.
Syntax	RC <i>character-replacement-list</i> [<i>column-reference</i>]
<i>character-replacement-list</i>	a quoted string of the form 'before=after' the <i>before</i> and <i>after</i> parts are strings of the same length
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> may potentially be modified
Example command	C P'@@@@' WORD M"1+ RC`() db=[]42`"
Source string	(AbCd)
Result string	[A2C4]
Remarks	<p>The relationship between characters on the left side of the = sign and the right side is positional. Notice the (is associated with [; the) is associate with] and so on. The <i>before</i> and <i>after</i> parts cannot themselves contain = signs when these are part of a single value as in 'before=after'.</p> <p>The <i>before</i> and <i>after</i> parts can also be specified as 'before'= 'after'. Example: M"1+ RC`123`='xyz`". When this is done, each part may optionally have its own string type code of C, T or X, and may contain embedded equal signs as data. A string value with a <i>string-pair separator</i> of ":" instead of "=" signifies a <i>case-conformant</i> change. See section 9, Case sensitivity and case conformance, for an explanation of how type codes on string pairs are handled.</p>

RL - Replace Left-most

Summary	Replace left-most instance of <i>first</i> string value with <i>second</i> string value
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy. Otherwise, codes > and < honored; codes <> and >< disregarded in favor of explicit string type codes.
Syntax	RL <i>string-values</i> [<i>column-reference</i>]
<i>String-values</i>	A quoted string of the form 'first=second' and may contain a string type code. The <i>first</i> and <i>second</i> parts can be of different lengths; the <i>second</i> part can be null, but the <i>first</i> part cannot. When <i>second</i> string is null, command acts to delete the <i>first</i> string. The <i>before</i> and <i>after</i> parts cannot contain = equal signs as data if specified in the single <i>string-values</i> form.
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> may potentially be modified
Example command	C P'===== WORD M"RL`pay=taxes`"

Source string	gross_pay,net_pay
Result string	gross_taxes,net_pay
Remarks	When characters to the left of = sign contain letters, see main article for explanation how the case of letters is handled. The <i>first</i> and <i>second</i> parts can be specified as a <i>string pair</i> ; example: M"RL`pay` = `taxes` " . When this is done, each part can have its own string type code of C, T or X, and can contain = equal signs as data. A string value with a <i>string-pair separator</i> of ":" instead of "=" signifies a <i>case-conformant</i> change. See section 9, <u>Case sensitivity and case conformance</u> , for an explanation of how type codes on string pairs are handled.

RP - Repeat the result string

Summary	Repeat the <i>result string</i> a specified number of times
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	RP [<i>replication-factor</i>]
<i>replication-factor</i>	A numeric value to indicate the number of times the <i>result string</i> is to be replicated; if omitted, a value of 1 is assumed. If specified as zero, no replication occurs.
Example command	C P"@@@ M"RP"
Source string	Abc
Result string	AbcAbc
Remarks	RP makes one or more copies of whatever is in the <i>result string</i> at the point RP is used. The <i>replication-factor</i> specifies the number of <i>additional</i> copies of the <i>result string</i> are made. Thus, after an RP2 command is issued, there will be 3 copies (2 additional ones) of the <i>result string</i> .

RR - Replace Right-most

Summary	Replace right-most instance of <i>first</i> string value with <i>second</i> string value
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy. Otherwise, codes > and < honored; codes <> and >< disregarded in favor of explicit string type codes.
Syntax	RR <i>string-values</i> [<i>column-reference</i>]
<i>String-values</i>	A quoted string of the form ' <i>first</i> = <i>second</i> ' and may contain a string type code. The <i>first</i> and <i>second</i> parts can be of different lengths; the <i>second</i> part can be null, but the <i>first</i> part cannot. When <i>second</i> string is null, command acts to delete the <i>first</i> string. The <i>before</i> and <i>after</i> parts cannot contain = equal signs as data if specified in the single <i>string-values</i> form.
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> may potentially be modified
Example command	C P'===== WORD M"RR`pay=taxes`"
Source string	gross_pay,net_pay
Result string	gross_pay,net_taxes
Remarks	When characters to the left of = sign contain letters, see main article for explanation how the case of letters is handled. The <i>first</i> and <i>second</i> parts can be specified as a <i>string</i>

	<i>pair</i> , example: M"RR`pay` = `taxes` " . When this is done, each part can have its own string type code of C, T or X, and can contain = equal signs as data. A string value with a <i>string-pair separator</i> of ":" instead of "=" signifies a <i>case-conformant</i> change. See section 9, <u>Case sensitivity and case conformance</u> , for an explanation of how type codes on string pairs are handled.
--	---

RV - Reverse the contents of the result string

Summary	Reverse the contents of the <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	RV [<i>column-reference</i>]
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is reversed
Example command	C P'@@@' M"\` 1+ `/` RV"
Source string	ABC
Result string	/CBA\
Remarks	If RV were the only command, the <i>result string</i> would be CBA , due to <i>auto-copy</i> . Because a reverse column reference can also reverse characters, the main use of RV is to reverse the <i>result string</i> (or part of it) after a complex value is built up from several commands.

SC - Set result string to Sentence Case

Summary	Sets contents of <i>result string</i> to Sentence Case, as of the point the sc command is issued
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	sc [<i>column-reference</i>]
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is set to sentence case
Example command	C P'@@@ @@@' M"1+ SC"
Source string	ONE two
Result string	One two
Remarks	See main help for a description (Primary and Line commands) of what Sentence Case means.

S* - Generate Sequence number in Decimal and Hex – SD / SX

Summary	SD - Convert sequence number to decimal and apply numeric formatting SX - Convert sequence number to hex and apply numeric formatting
Command type	Modifying commands; will not Auto-Copy
Case mod codes	Codes > and < honored; codes <> and >< treated as >
Syntax	{ SD SX } [<i>format</i>] [<i>sign</i>] [<i>calculation-operand</i>]

format	<p>[<i>w</i>] <i>t</i></p> <p>w - width of result number; <i>w</i> is required for format types F and Z, optional for type V. If omitted for type V, <i>w</i> is assumed to be 1. If omitted for types F and Z, <i>w</i> is taken as minimum size needed to format the number without truncation.</p> <p>t - format type code</p> <ul style="list-style-type: none"> • format F is fixed-width with zero-fill or truncation on left • format Z is fixed-width with zero-suppression or truncation on left • format V is variable width with zero-fill on left; for V code, <i>w</i> means minimum width and a <i>w</i> value of 0 is allowed
sign	If sign code s is present, any leading + or – sign found during processing will be retained and copied to <i>result string</i> . If sign code s is absent, any leading + or – signs are ignored. Any spaces after the leading + or – sign are considered to be part of the sign. s
column-reference	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C R' [A-Za-z0-9_]+' WORD M"1+ '_' SD 5F'*2 +1 ''
Source string	record_counter
Sequence number	25
Result string	record_counter_00501
Remarks	The value 5 is the format length . See Working with Mapping Strings to Reorder, Reformat and Transform Data for a description of how the casing of hex digits > 9 is rendered.

SQ - Insert Single Quote into result string

Summary	Insert Single Quote into result string
Command type	Modifying command; will not Auto-Copy
Case mod codes	Disregarded
Syntax	SQ [<i>replication-factor</i>]
<i>replication-factor</i>	Decimal value indicating the number of single quotes inserted. If omitted, a one quote is inserted. If specified as zero, no quotes are inserted.
Example command	C P"@@@@" M'SQ 1+ SQ'
Source string	ABCDE
Result string	'ABCDE'
Remarks	This command is useful to insert one or more single quotes when the outer M string is itself enclosed in single quotes. Since this command does not auto-copy, you must explicitly specify other values you want in the result string, as is done with the 1+ code in the example.

SV - Specify a Set Variable as an external mapping string item

Summary	Anamed SPFLite SET Variable is used as a mapping item
---------	---

Command type	Special case; does not copy or modify <i>result string</i>
Case mod codes	Disregarded
Syntax	SV <i>symbol-name</i>
<i>symbol-name</i>	A quoted-string value naming an SPFLite SET variable.
Example command	C P"##=##=##" M"SV'ABC'"
Contents of MAPSTR. SV. ABC	RC'*'= /
Source string	12*34*56
Result string	12/34/56
Remarks	When you specify a <i>symbol-name</i> , a prefix is added to the name you supply. If you issue a mapping command of SV'ABC' , the SPFLite SET variable that is searched for is MAPSTR. SV. ABC . The mapping commands retrieved from the SET variable are substituted at the point where they are referenced in your M string.

TC - Set result string to Title Case

Summary	Sets contents of <i>result string</i> to Title Case, as of the point the TC command is issued
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	TC [<i>column-reference</i>]
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is set to title case
Example command	C P'@@@ @@@' M"1+ TC"
Source string	 ONE two
Result string	 One Two
Remarks	See main help for a description (Primary and Line commands) of what Title Case means.

T - Trim the result string on both sides

Summary	Remove <i>pad characters</i> from both the left and right sides of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	T [<i>pad-character</i>] [<i>maximum-trim</i>]
<i>pad-character</i>	If omitted, the pad character being removed is a single blank
<i>maximum-trim</i>	Maximum number of pad characters to be removed from each side. If omitted, all possible padding characters are removed.
Example command	C P"====@@@@====' M"1+ T`*`"
Source string	 ****AAcc***
Result string	 AAcc
Remarks	To remove just 1 asterisk on each side and produce **AAcc** , use M"1+ T`*`1"

TL - Trim the result string on Left side

Summary	Remove <i>pad characters</i> from left side of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	TL [<i>pad-character</i>] [<i>maximum-trim</i>]
<i>pad-character</i>	If omitted, the pad character being removed is a single blank
<i>maximum-trim</i>	Maximum number of pad characters to be removed from left side. If omitted, all possible padding characters are removed from left side.
Example command	C P'====@@@@====' M"1+ TL`*`"
Source string	****AAcc***
Result string	AAcc***
Remarks	To remove just 1 asterisk on the left and produce **AAcc*** , use M"1+ TL`*`1"

TR - Trim the result string on Right side

Summary	Remove <i>pad characters</i> from right side of <i>result string</i>
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	TR [<i>pad-character</i>] [<i>maximum-trim</i>]
<i>pad-character</i>	If omitted, the pad character being removed is a single blank
<i>maximum-trim</i>	Maximum number of pad characters to be removed from right side. If omitted, all possible padding characters are removed from right side.
Example command	C P'====@@@@====' M"1+ TR`*`"
Source string	****AAcc***
Result string	****AAcc
Remarks	To remove just 1 asterisk on the right and produce ***AAcc** , use M"1+ TR`*`1"

UC - Set result string to Upper Case

Summary	Sets contents of <i>result string</i> to Upper Case, as of the point the UC command is issued
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	UC [<i>column-reference</i>]
<i>column-reference</i>	selects columns from <i>result string</i> only , whether <i>dot notation</i> used or not if omitted, the entire <i>result string</i> is set to upper case
Example command	C P'@@@@' M"1+ UC"
Source string	AAcc
Result string	AAACC

Remarks	
---------	--

X - Exchange strings

Summary	If <i>source string</i> contains one of two values, exchange first with second or vice-versa
Command type	Copying command; will Auto-Reference
Case mod codes	Honored during auto-copy. Otherwise, codes > and < honored; codes <> and >< disregarded in favor of explicit string type codes.
Syntax	X exchange-pair column-reference
exchange-pair	' first=second ' or ' first'='second '
column-reference	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C R' [a-z]+' M"X`one=five`1+"
Source string	one
Result string	five
Remarks	<p>If found <i>source string</i> is one the value five is appended to <i>result string</i>.</p> <p>If found <i>source string</i> is five the value one is appended to <i>result string</i>.</p> <p>If found <i>source string</i> is eight the value eight is appended to <i>result string</i>.</p> <p>Use <i>exchange-pair</i> format 'first'='second' if first or second contain embedded = equal signs. When characters of first string contain letters, see Working with Mapping Strings article for an explanation how the case of letters and string type-codes are handled.</p> <p>An <i>exchange-pair</i> with a trailing type code of T signifies a <i>case-conformant</i> change.</p>

XA - Perform Hex to Ansi character conversion

Summary	Convert encoded hex digits to their corresponding Ansi character data
Command type	Copying command; will Auto-Reference
Case mod codes	All codes honored
Syntax	XA [bypass-string] column-reference
column-reference	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
bypass-string	Characters in <i>bypass-string</i> are not converted, but are copied as-is to the <i>result string</i>
Example command	C P'====\====' M"XA`-`1+"
Source string	3132-4C62
Result string	12-Lb
Remarks	Note '-' was not converted but copied. After any bypassed characters are discounted, what remains must be an even number of hex-digit characters. Hex digits > 9 are always recognized in case-insensitive mode.

XD - Perform Hex to Decimal numeric conversion

Summary	Convert hex data to decimal and apply numeric formatting
---------	--

Command type	Copying command; will Auto-Reference
Case mod codes	Disregarded
Syntax	{ XD <i>left-to-right-cmd</i> <i>right-to-left-cmd</i> } [<i>format</i>] [<i>sign</i>] <i>column-reference</i>
<i>left-to-right-cmd</i>	nXD
<i>right-to-left-cmd</i>	xDn
<i>n</i>	A value from 1 to 9 , indicating the relative position within the selected columns where the number is found after scanning for it, where 1 means first, and 9 means ninth. When n is used as nXD , the scan for the numeric value is done in a left-to-right manner, and when n is used as xDn , the scan for the numeric value is done in a right-to-left manner. If n value is omitted, the XD command expects the full selected column range to be a single hex number and no scan is performed to locate the value.
<i>format</i>	[w] t w - width of result number; w is required for format types F and Z , optional for type V If omitted for type V , w is assumed to be 1 . If omitted for types F and Z , w is taken as the same width of the original number as found in the <i>source string</i> . t - format type code format F is fixed-width with zero-fill or truncation on left format Z is fixed-width with zero-suppression or truncation on left format V is variable width with zero-fill on left; for V code, w means minimum width and a w value of 0 is allowed
<i>sign</i>	If sign code S is present, any leading + or - sign found during processing will be retained and copied to <i>result string</i> . If sign code s is absent, any leading + or - signs are ignored. Any spaces after the leading + or - sign are considered to be part of the sign. If sign code C is present, it is treated like sign code S is, with the following addition: If a number contains a 0x or 0X prefix (used in C and similar languages), that prefix is considered to be part of the sign, and not part of the number. The reason this is needed is that without this, a value like 0x12 might appear to be two numbers, 0 and 12 , separated by a delimiter of " x ".
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C P'==' WORD M"XD3Z1+"
Source string	 0F1
Result string	 °15
Remarks	Hex digits > 9 are always recognized in case-insensitive mode. The value 3 is the format length . To produce 015 use M"XD3F1+" The ° symbol represents one blank.

XE - Perform Hex to EBCDIC character conversion

Summary	Convert encoded hex digits to their corresponding EBCDIC character data
---------	---

Command type	Copying command; will Auto-Reference
Case mod codes	All codes honored
Syntax	XE [<i>bypass-string</i>] <i>column-reference</i>
<i>column-reference</i>	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
<i>bypass-string</i>	Characters in <i>bypass-string</i> are not converted, but are copied as-is to the <i>result string</i>
Example command	C R' [-A-Fa-f0-9]+ WORD M"XE`-`1+
Source string	 F1F2-D382
Result string	 12-Lb
Remarks	Note ‘-’ was not converted but copied. After any bypassed characters are discounted, what remains must be an even number of hex-digit characters. Hex digits > 9 are always recognized in case-insensitive mode. The EBCDIC translation used will be the standard 1140 code page, without regard to the SOURCE code page setting.

XX - Perform Hex to Hex numeric conversion

Summary	Copy hex data and apply numeric formatting
Command type	Copying command; will Auto-Reference
Case mod codes	Codes > and < honored; codes <> and >< treated as >
Syntax	{ xx <i>left-to-right-cmd</i> <i>right-to-left-cmd</i> } [<i>format</i>] [<i>sign</i>] <i>column-reference</i>
<i>left-to-right-cmd</i>	nxx
<i>right-to-left-cmd</i>	xnx
n	A value from 1 to 9 , indicating the relative position within the selected columns where the number is found after scanning for it, where 1 means first, and 9 means ninth. When n is used as nxx , the scan for the numeric value is done in a left-to-right manner, and when n is used as xnx , the scan for the numeric value is done in a right-to-left manner. If n value is omitted, the xx command expects the full selected column range to be a single hex number and no scan is performed to locate the value.
<i>format</i>	[w] t w - width of result number; w is required for format types F and Z , optional for type V If omitted for type V , w is assumed to be 1 . If omitted for types F and Z , w is taken as the same width of the original number as found in the <i>source string</i> . t - format type code format F is fixed-width with zero-fill or truncation on left format Z is fixed-width with zero-suppression or truncation on left format V is variable width with zero-fill on left; for V code, w means minimum width and a w value of 0 is allowed
<i>sign</i>	If sign code S is present, any leading + or – sign found during processing will be retained and copied to <i>result string</i> . If sign code S is absent, any leading + or – signs are ignored. Any spaces after the leading + or – sign are considered to be part of the sign. If sign code C is present, it is treated like sign code S is, with the following addition: If a number contains a 0x or ox prefix (used in C and similar languages), that prefix is

	considered to be part of the sign, and not part of the number. The reason this is needed is that without this, a value like 0x12 might appear to be two numbers, 0 and 12, separated by a delimiter of “x”.
column-reference	selects columns from <i>source string</i> , or from <i>result string</i> if <i>dot notation</i> is used
Example command	C P'== WORD M"XX3Z1+"
Source string	0F
Result string	°0F
Remarks	<p>Hex digits > 9 are always recognized in case-insensitive mode.</p> <p>The value 3 is the format length.</p> <p>To produce 00F use M"XX3F1+"</p> <p>The ° symbol represents one blank.</p>

Z - Zero-suppress the result field

Summary	Zero-suppress a formatted decimal number
Command type	Modifying command; will Auto-Copy
Case mod codes	Honored during auto-copy, otherwise disregarded
Syntax	z [ignore-characters] [minimum-length]
ignore-characters	one or more non-significant leading characters to be ignored/suppressed; typically will include comma (U.S. format) or dot (European format)
minimum-length	the right-most number of characters that will not be suppressed, regardless of content; if omitted, there is no minimum length, leading to possible suppression of entire field
Example command	C P"##\$##\$##" WORD M"1+ z', '3"
Source string	0,001.23
Result string	°°°°°1.23
Remarks	<p>The ° symbol represents one blank.</p> <p>Non-significant leading zeros and spaces are always ignored/suppressed</p> <p>A value of 0,000.23 would result in °°°°°.23 </p> <p>A value of 0,000.03 would result in °°°°°.03 </p>

Working with Sequence Numbering

Contents of Article

[Related Commands](#)

[Introduction](#)

[Differences Between ISPF Numbering and the SPFLite Support](#)

[Protect yourself from sequence numbering mistakes](#)

[Dealing with the two use case of sequence numbering](#)

[Understanding sequence numbering types](#)

[Dealing with sequence numbers that get in your way](#)

[Automatic sequence renumbering during SAVE and END](#)

[Determining the value of the new sequence numbers for RENUM](#)

[Determining the value of the new sequence numbers for NUMBER](#)

[Synchronizing sequence numbers with SPFLite](#)

Related Commands

See the following commands for more detailed information about the SPFLite Sequence Numbering feature.

[AUTONUM](#) Set Number lines automatically on SAVE

[NUMBER](#) Verify and correct sequence numbers if needed

[NUMTYPE](#) Define Numbering Style to be used

[RENUMBER](#) Renumber all sequence numbers

[UNNUMBER](#) Remove sequence numbers

Introduction

ISPF users have the ability to add, renumber and remove sequence numbers in their data, but this facility was never migrated to SPFLite since numbering of PC files is quite rare. However with the increased use of IBM mainframe simulators (like Hercules) there is a place and need for such numbering. Starting in SPFLite version 8.5, you can now perform sequence-numbering actions using commands that are highly compatible with ISPF, and which also provide the additional flexibility you have come to expect from SPFLite.

Because the SPFLite Sequence Numbering feature is new, users are also encouraged to report any issues they encounter with it, to help us ensure it works as intended and meets your needs.

Differences Between ISPF Numbering and the SPFLite Support

There is a significant difference between the way ISPF and SPFLite manage files containing sequence numbers.

Once a valid set of sequence numbers exist, ISPF will maintain sequence numbers as lines are inserted, copied and moved around within the file. This is done dynamically as you edit.

SPFLite does **not** perform this function, line numbers are only set when a [NUMBER](#) or [RENUM](#) command is performed, not dynamically during editing.

Differences can be summarized as:

- No dynamic numbering support from SPFLite. As a result, the concepts of **NUMBER ON** and **NUMBER OFF** do not exist, and these keywords are not supported by the Numbering commands.
- Since the sequence number fields are simply treated as part of the normal data in SPFLite, the former

ISPF concepts of **NUMBER DISPLAY** and **NUMBER NODISPLAY** do not exist. Therefore the **DISPLAY** and **NODISPLAY** keywords are not supported by the Numbering commands.

- **AUTONUM** with no operands in ISPF is treated as **AUTONUM ON**. This is not true in SPFLite, you must explicitly enter **ON** or **OFF** as this is the normal SPFLite convention for On/Off settings
- ISPF supports a **LEVEL** concept for the sequence field. SPFLite does not support this.
- ISPF supports files which have both COBOL **and** Standard numbering. This was rarely, if ever, used. SPFLite will only allow one numbering style to be defined or applied at any given time.
- ISPF links the numbering style to the file **type**. SPFLite maintains the numbering style within the **Edit Profile**, which may or may not be the same name as the file extension.

The ISPF keywords **NOSTD**, **NOCOBOL** and **NOCOB** are also unsupported as they were used in ISPF to manage sequencing of a file when both STD and COBOL numbering types are in effect at the same time.

Protect yourself from sequence numbering mistakes

As with any new facility, if you choose to try the Numbering facilities, be cautious until you become familiar with it. Unlike many other commands, the Numbering commands have the potential to overlay and destroy data unexpectedly if you specify things incorrectly. Make sure you know how to use [UNDO](#) to reverse the effects of any editing action. Or simply use [CANCEL](#) to exit your edit session, resolve the problem, and try again. Finally, it is recommended that you experiment with test files first, to help you understand how these commands operate.

Lastly, note that the sequence numbering commands consider a non-text file type as "exempt" from being numbered or renumbered, and will reject a numbering command from being applied to it. The file extensions of non-text files are listed in the File Manager tab of the SPFLite Global Options GUI.

If for any reason you believe the numbering commands are not operating in compliance with the documentation, please let us know on the SPFLite user forum.

Understanding sequence numbering types

Unlike ISPF, which only knew about two "types" of numbering: "Standard" numbering, and COBOL numbering, SPFLite has not tied itself to any preconceived ideas about where in a text line the sequence numbers should appear, nor their length.

SPFLite has introduced a new numbering command - **NUMTYPE** - to allow you to describe how **you** want to place the sequence number field. Once defined by the **NUMTYPE** command, the **NUMBER** / **RENUM** / **UNNUMBER** commands all perform their function under control of those **NUMTYPE** specifications.

So just what options can be specified?:

- Sequence number fields may be from 4 to 8 characters in length.
- The field may be placed anywhere within the text record, **you** specify the start and end columns.
 - If a text line is shorter than needed to contain the field, it will be automatically padded with sufficient blanks.
 - Fields may even be specified using relative columns **from the right end of the line**.
- The numbering function can be requested to truncate the line following the sequence field.
- You may request the sequence field be set equal to the SPFLite line number itself rather than an artificial incrementing number.

Note: It is quite possible for you to specify a **NUMTYPE** specification that is **spectacularly** harmful. SPFLite will slavishly follow your orders. E.G. If you **really** specify putting sequence numbers right in columns 20-27 of your source file, SPFLite will not question it.

BE CAREFUL!

However, to help you, SPFLite doesn't just ignore the old ISPF standards. The **NUMTYPE** command accepts 3 shortcuts to accommodate this:

NUMTYPE COB	Will setup a COBOL sequence field in columns 1 - 6.
NUMTYPE STD	Will setup a standard sequence field in columns 73 - 80
NUMTYPE IBM	Will setup a standard sequence field in columns 73 - 80 and request truncation after column 80.

Dealing with sequence numbers that get in your way

ISPF integrates the presence of sequence numbers into its editor logic. SPFLite does not. As a result, there is nothing "magic" or "special" about these numbers. They are just strings of digit characters that exist in your data.

In particular, ISPF maintains the screen locations for displayed sequence numbers if you use the LEFT or RIGHT primary commands (normally mapped to F10 and F11) so they don't shift on the screen. However, in SPFLite, LEFT and RIGHT will include those numbers as simply another part of your user data, and they will get shifted along with everything else.

As you edit your file, if you have sequence numbers, say in columns 73-80, these will get "pulled" or "pushed" as you use the DEL key or type data in Insert Mode. That may cause problems, including making your data invalid for the purposes you intend. For instance, it could create syntax errors in a source program.

To resolve such problems, you can UNNUMBER your file, perform your changes and then NUMBER the file..

Automatic sequence renumbering during SAVE and END

SPFLite supports the **AUTONUM** ability. Simply issue **AUTONUM ON** or **AUTONUM OFF** to control this.

To use automatic sequence numbering a valid **NUMTYPE** specification must exist.

Determining the value of the new sequence numbers for RENUM

The RENUM commands will completely assign new sequence numbers. The starting number and increment are based on the number of lines in the file, and the specified size of the sequence number field. The following table summarizes this.

File Size	Sequence # Size	Starting Number	Increment
Under 1000	4	0010	10
	5	00100	100
	6	000100	100
	7	0000100	100
	8	00000100	100
Under 10000	4	0001	1
	5	00010	10
	6	000100	100
	7	0000100	100
	8	00000100	100
Under 100000	4	0001	1
	5	00001	1

	6	000010	10
	7	0000100	100
	8	00000100	100
Under 1000000	4	0001	1
	5	00001	1
	6	000001	1
	7	0000010	10
	8	00000100	100

Determining the value of the new sequence numbers for NUMBER

The **NUMBER** command is different from **RENUM**, in that it will only *conditionally* renumber a file, and then only where needed. Its primary purpose is to **verify** that a file's sequence numbers are present, valid and in ascending order, and only acts to correct the file if those conditions are not met.

If **NUMBER** finds the file already contains a valid set of sequence numbers, **no action** will be performed.

Otherwise, **NUMBER** attempts to correct out-of-sequence numbers in a way that causes the fewest changes to the current sequence numbers that are already present. As such, you may end up with lines numbered with varying increments between them. This is not an error, merely the result of keeping the current valid numbers and **fitting in** the corrections between them. If your file is severely out of sequence, **NUMBER** could end up abandoning most or all of the original sequence numbers.

Synchronizing sequence numbers with SPFLite

ISPF provides two basic numbering options. **NUMBER** will attempt to change as little as possible, keeping existing sequence numbers when it can. **RENUM** changes every line, creating new numbers that have uniform values.

SPFLite provides a third option, using the **NUMTYPE** option **REAL**. Specifying **REAL** on **NUMTYPE** will cause the **RENUM** command to use the line numbers known to SPFLite as the numeric values of the new sequence number field, rather than trying to retain any existing numbers or calculating new values for them.

Using the SPFTest Program

The use of Regular Expressions, Mapping Strings and Mapping Calculation Operands involves the use of command operands and command codes that can be cryptic and unfamiliar, especially if you don't use them frequently. As extensively as these features are documented, they could fail or could produce results you may not understand or did not expect. In many cases, you may only observe these features seemingly doing nothing, with no explanation as to what went wrong.

The SPFTest program was designed to provide you with a 'scratch pad' for experimenting with these features, and immediately see results, without having to set up test data in an SPFLite edit session. Since these tests don't involve actual data, you can't harm anything by running these tests, and you can quickly repeat tests without having to issue SPFLite primary commands.

The SPFTest program can be accessed in two ways:

- A shortcut is available in the SPFLite Start menu folder
- Within SPFLite, you may enter a TEST command, which will start a copy of the SPFTest program. The TEST command may optionally have an operand of:
 - **REGEX** - to open SPFTest with the RegEx tab active.
 - **MAP** or **MAPPING** - to open SPFTest with the Mapping tab active
 - **CALC** - to open SPFTest with the Calculator tab active

The SPFTest program, once started, runs independently of SPFLite itself, and is not dependent on it, so you can switch between the two without one program affecting or interfering with the other.

There are three complex operand types in SPFLite which the SPFTest program provides interfaces to experiment with.

Regular Expression Strings

This is the interface for testing Regular Expression strings. The top half of the screen contains three input fields.

Default Case

Regex character matching can be defaulted to Case Sensitive or Case Insensitive. This input field is where you can specify the default you wish. Enter for case sensitive or **T** for Case insensitive. Note: within a Regex string, this default can be overridden with a special code. See the [Specifying a Regular Expression](#) article for more information.

Some aspects of regular expressions will behave differently, based on what the current CASE setting is within SPFLite. This field is used to simulate whether CASE C or CASE T is in effect when you execute the given regular expression.

Source string

This is the text string against which the Regex test will be performed. Think of it as being a sample line from the Edit file.

Regex Expression

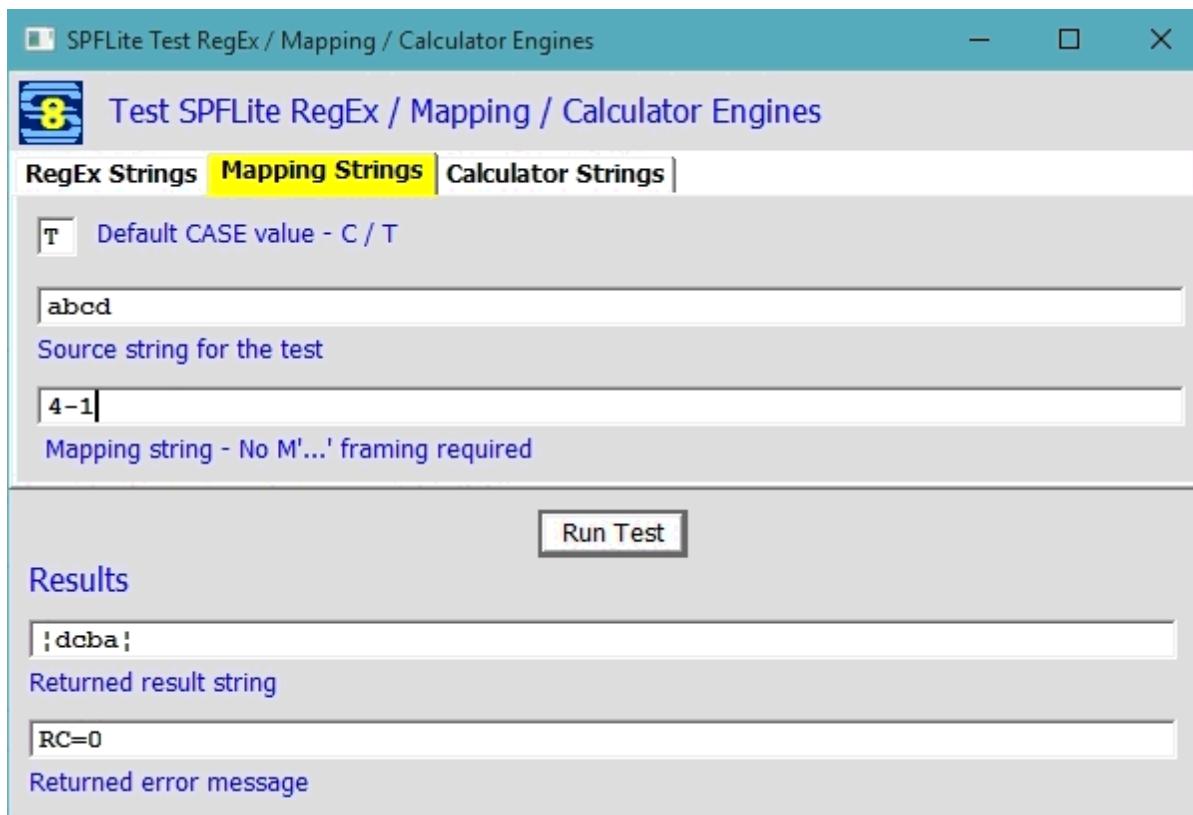
The Regex expression to be tested. Note: In an actual SPFLite command, this would be included within the quotes of an R-Type literal. In SPFTest, do not specify this string with the R"xxx" format. Leave the R and quote marks off, and only enter the regular expression itself.

Complete these three fields and Press Enter, or click the **Run Test** button. The results will be displayed in the bottom half of the screen. If a matching string is found, it will be displayed in the upper box, surrounded by | characters. (like the |ghi| in the image above) The lower box will contain information about the located match string, or it will contain any error messages if the test is unsuccessful.

Starting in SPFLite release 8.3, a new version of Regular Expression syntax is now accepted, based on the well-known, open-source PCRE regular expression engine. This RegEx engine is more powerful than what was previously available with SPFLite, but you should be mindful that there are a few, minor inconsistencies between the two systems. This should not cause you any difficulties, but you should be aware of how they differ.

For a full explanation of Regular Expressions, see [Specifying a Regular Expression](#).

Mapping Strings



This is the interface for testing Mapping strings. The top half of the screen contains three input fields.

Default Case

Certain mapping subcommands are sensitive to character case. The default handling can be set to Case Sensitive or Case Insensitive. This input field is where you can specify the default you wish. Enter **C** for case sensitive or **T** for Case insensitive.

Some aspects of mapping commands will behave differently, based on what the current CASE setting is within SPFLite. This field is used to simulate whether CASE C or CASE T is in effect when you execute the given mapping string.

Source string

This is the text string against which the Mapping test will be performed. Think of it as being the string which was located by the search operand of a CHANGE command.

Mapping String

The Mapping expression to be tested. Note: in an actual SPFLite command, this would be included within the quotes of an M-Type literal. In SPFTest, do not specify this string with the M"xxx" format. Leave the M and quote marks off, and only enter the mapping string itself.

Complete these three fields and Press Enter or click the Run Test button. The results will be displayed in the bottom half of the screen. If a matching string is found, it will be displayed in the upper box, surrounded by | characters. (like the |dcba| in the image above) The lower box will contain information about the located match string, or it will contain any error messages if the test is unsuccessful.

This description is not intended to explain Mapping strings in detail. For that please see [Full Mapping String](#)

Calculator Strings

SPFLite Test RegEx / Mapping / Calculator Engines

Test SPFLite RegEx / Mapping / Calculator Engines

RegEx Strings | Mapping Strings | **Calculator Strings**

1 Initial value for the S variable Increment value before each test?

123 Initial value for the R and X variables

S*4 Calculation string to be evaluated

Run Test

Results

4 Hex --> .0000000000000004

Returned result string

RC=0

Returned error message

This is the interface for testing Calculator strings. Calculator strings are a part of Mapping strings, but are of sufficient complexity to be separated out here for experimentation. The top half of the screen contains three input fields.

Initial S variable value	The initial value that you wish to be established for the Calculator S variable. The S variable is intended to represent a 'sequence number' or 'serial number' for each instance of a string found by a CHANGE ALL primary command that involves mapping strings. When you issue such a primary command, the first string found has a sequence number of 1, the second has 2, and so on. The value here is used to simulate such sequence numbers, if you have a calculator expression that has a reference to variable S.
Increment value before test	If this option is selected, the value in the Initial box will be incremented by one before each test. You might use this if you were testing a calculator expression that has a reference to variable S.
Initial value for R and X variables	The initial value to be established for the R and X variables.
Calculation string	The actual calculation string to be evaluated. This string will become a portion of a full Mapping string. Calculator expression operands are permitted on the Mapping String commands DD, DX, XD, XX, SD and SX.

Complete these three fields and Press Enter or click the **Run Test** button.

The results will be displayed as a decimal number and its Hex equivalent in the bottom half of the screen. Note that mapping string calculation operands use a leading dot to signify a hex number.

The lower box will contain the result including any generated error messages.

For a complete description of mapping string calculation operations, see [Mapping String Calculation Operands](#).

Shifting Data

Contents of Article

- [Introduction](#)
- [Differences between Column Shift and Data Shift line commands](#)
- [Indent Shift - a new kind of Column Shift line command](#)
- [Data-Shifting Insert-Mode function](#)
- [Data-Delete function](#)
- [Data-Backspace and Data-Delete-Mark](#)

Introduction

When you edit data, the editor automatically shifts characters on a line to the left or right to accommodate insertions or deletions. This shifting can be either implicit or explicit. Implicit shifts occur when **INS** mode is in effect when you press the Insert key, deleting characters with the **DEL** key, or when the **CHANGE** command string-2 length is different from the string-1 length.

There are three new facilities available for shifting of data. These are **Data-Shifting Insert Mode**, and the **Data Delete** function, described at the end of this section, and the **Indent Shift** line commands described here.

Explicit shifts occur when you use the following line commands:

- (or ((Column Shift Left
-) or)) Column Shift Right
- < or << Data Shift Left
- > or >> Data Shift Right
- [or [[Indent Shift Left
-] or]] Indent Shift Right

Two columns is the standard default for shift operations. This default can be changed to any number of columns desired (in the range of 1 through 999 columns) by a General Options setting.

When shifting a block of lines more or less than the default, enter the amount on the first or last line of the block. If you enter it in both places, the line shifts only if both amounts are the same.

Differences between Column Shift and Data Shift line commands

There are some significant differences between Column shifts and Data Shifts. Shifting occurs within the column boundaries (either the default - the whole line) or those specified with the **BOUNDS** command or the **BNDS** line command.

Column Shift moves all characters within the bounds without altering their relative spacing. Characters shifted past the bounds are deleted. That is, blanks are inserted at the bound *from* which the characters are being shifted, and the characters are deleted at the *opposite* bound. So, this shift is called a destructive shift because information shifts within column boundaries without regard to its contents, and can result in the loss of data with no error being noted.

Data Shift moves characters non-destructively within the BOUNDS. The end-result of a data shift is a change in the widths of one or more spans of blanks; nonblank data will not be changed or deleted.

For data shift left attempts that exceed the current Left BOUNDS setting, text movement stops at the left bound.

Note: The ISPF editor marks lines with ==ERR> flags when a data shift operation can only be partially completed, either because of the data contents of the line or the current location of the BOUNDS. SPFLite recognizes the same conditions, but currently does not support the display of ==ERR> flags. Instead, it will display a message if there were any lines that could not completely shifted. The message will say "Data shifting incomplete" and will report the number of lines for which the full shifting amount could not be applied.

Data shifting works by moving contiguous spans of nonblank characters as a unit, and 'pushing' adjacent spans of characters as needed to make sure there is no data loss. When spans of nonblank characters are separated by only one space, that space is never collapsed (deleted); that way, the spans of nonblank data are never 'run together'.

For a data shift right, shifting can occur up to and including the right bound column. If the right bound is MAX, right-shifting the data line may result in making it longer.

For a data shift left, shifting can occur up to but not including the right bound column.

If left BOUND column of the data line is nonblank, it defines a "label" field, which is locked at that position and is not moved during either a left or right data shift.

Note: The Data Shift commands are intended to allow for non-destructive shifting of the kind of data that appears in typical programming-language statements, by applying some general rules. These rules require a certain amount of parsing and analysis of your data. The support in SPFLite for Data Shifting commands has been extensively tested, and great efforts were made to ensure that it handles data in an ISPF compatible manner. However, because these general rules may or may not be applicable or useful in your particular situation, you should inspect the results of a Data Shift to ensure you obtained the results you intended, until you are familiar with how this feature operates.

Example:

```
=BNDS>      <                                >
000010      Word1      Word2      Word3      Wordn
```

If a Data Shift Right command of >10 is entered on line 00010 the result would be

```
=BNDS>      <                                >
000010      Word1      Word2      Word3      Wordn
```

As can be seen, Word1 and Wordn have been left alone, and only the other data shifted.

Note: Data Shift line commands have the following features:

- Spaces contained inside of strings enclosed in ' single quotes or " double quotes are considered 'protected' and will not be compressed or expanded as a result of data shifting. For this 'protection' to apply, the string must be properly closed. That is, there must a close-quote of the same kind that begins the string value. An individual quote (such as an apostrophe in a word like **don't**) is ignored for purposes of protecting spaces. There is presently no support for treating ` accent quotes in the same way; accent characters are ordinary data.
- IBM ISPF documentation states that both ' single quotes and " double quotes are supposed to be handled the same way, but tests show that their string handling is not consistent with their documentation, and is release-dependent. SPFLite fully supports both quote types in this regard.
- When you perform data shifting on data that has colors, the colors are shifted as well, so that the underlying colors of your nonblank data will not change.
- If you issue a data shift command for a line that has colors, you need to be aware that **spaces can have colors as well**. Even though a space has no visible graphic symbol, there is still an underlying "color

attribute byte" just like any other data character on a line. **Colorized spaces may or may not be visible, depending on the background highlight color you choose for it in the Screen Options dialog.** In most cases, this is not usually important, but if you issue commands like **FIND "ABC DEF" RED**, the string will only be found if it is **entirely** RED, including the blank in the middle. When a Data Shift command moves spans of nonblank characters around on a line, existing spans of spaces can get larger or smaller. If these spans of spaces have color attributes, the attributes for the existing span of blanks are copied, as much as will fit, into the new span of blanks, if equal or shorter in length. If the new span is larger because blank data characters were inserted, the color attribute of the inserted blanks will be the same as the rightmost blank of the existing span of blanks (the rightmost color attribute is propagated). In the FIND example, if you had a string of "**ABC** **DEF**" with the blank in between having an attribute of RED, and it became "**ABC** **DEF**", all the new blanks would also be RED, and everything should work fine. However, if for some reason you had a string like "**ABC** **DEF**" in which the inner blanks had mixed color attributes, the color attributes are propagated on the right (the blue blanks are extended) or are truncated on the right. This may or may not be the results you intended. If your data has unusual color attributes, you may wish to avoid using data shifting on such data, or verify or re-establish the colors after the data has been modified by a data shifting operation.

- In order to properly handle quoted strings in the C language and similar syntax, a single or double quote preceded by a \ backslash can be treated as ordinary data and not as a quote. This is an extension to ISPF, which does not recognize escapes in string data. Support for recognizing \ backslash escapes is an **experimental feature**, enabled by the use of the SPFLite SET symbol **OPT.DS.ESCAPE**. If this SET symbol is defined as 1, backslash-escaped quotes will be bypassed; otherwise backslash characters will be treated as ordinary data. **Because this is an experimental feature, it is subject to change (or even removal) in future releases.**
- By default, spans of blanks are never compressed shorter than one blank. As an **experimental feature** it is possible to make the default minimum size of the space between nonblank spans larger, by the use of the SPFLite SET symbol **OPT.DS.MINSIZE**. If you define this, the symbol can have a value from 1 to 9. For example, a COBOL programmer might want to shift items in the Data Division using a default MINSIZE of 2, so that level numbers are separated from data names by two spaces. If you define this value as greater than 1, and a data shift operation would "push" one span of nonblanks next to another, and the space between them was already less than the defined minimum, the spacing is left as is rather than being increased to the minimum. For example, if you set OPT.DS.MINSIZE to 2, and do a right data shift of 10, suppose you had string like "**ABC DEF**" that was pushed right. Since the space between ABC and DEF is just one blank, it stays that way and is not expanded to 2, even though the "minimum" size was set to 2. The MINSIZE value only applies to existing spans that were already larger than the minimum. **Because this is an experimental feature, it is subject to change (or even removal) in future releases.**

Indent Shift - a new kind of Column Shift line command

SPFLite supports the *Indent Shift* facility. In terms of how data interacts with Bounds, as described above, an Indent Shift may be thought of as a specialized form of Column Shift. When there is no **n** value on an Indent Shift line command, the number of columns shifted is the same as the default shift columns value that appears in the General Options dialog. So, a simple **]** command with no **n** value works the same way as **)** does. And, as always, a command like **)n** overrides the default, so if the default is 4 and you say **)3** you indent 3 columns.

However, when a command like **]n** is used, the **n** value is **not** an override. Instead, the Indent Shift takes the established default number of columns and *multiplies it by n*.

So, if the default columns was 4, and you issued a **]3** line command, the number of columns shifted would be **4 x 3 = 12**. Suppose you are entering data, and wanted it formatted so that there were indentations at 4-column boundaries. Now, if you wanted to indent by 3 "indentation levels" you could use **]3** rather than **)12**.

Using Indent Shifts, you no longer have to think in terms of *columns*, but in terms of the number of *indentation levels* you are working with.

As an added convenience, because the Indent Shift line commands use the [and] bracket keys (which are non-shifted keys on many keyboards) these commands will be a little easier to type than (and) for most users.

Data-Shifting Insert-Mode function

There is a close relationship between SPFLite's standard Insert Mode and the) Shift Right line command. For example, if a line command of)4 is issued, four blanks are inserted at the beginning of the line, which is the same thing that would happen if Insert Mode is enabled, then the cursor is moved to the beginning of the line, and then finally the spacebar is pressed four times.

It is desirable to also have a relationship between an "insert mode" and the > Shift Right Data line command. It is now possible with the Data Shifting Insert Mode, enabled by the [\(DataInsert\)](#) keyboard function.

A suggested mapping for this key is Ctrl-Insert or Shift-Ctrl-Insert.

By the way, it's technically possible to map the Shift-Insert key also, but that is not recommended. If you are entering capital letters by holding down the shift key and then press Insert, you probably just want to enter normal Insert Mode and not do something special.

The [\(DataInsert\)](#) function will put the editor into Data Shifting Insert Mode, somewhat similar to the INS mode that occurs when the Insert key is pressed. Data Shifting Insert Mode allows data to be inserted into lines in a way similar to the way that the Data Shift line command > operates. Because of this, if you are inserting data that is formatted into columns, [\(DataInsert\)](#) will not disturb the column alignment, as long as two or more blanks separate the columns.

Looking at the example below, suppose we want to insert some data before the commas on each line, and we do not want to disturb the column containing CCC but want to keep it aligned. It may be possible to do this with **BOUNDS**, but **BOUNDS** can be inconvenient to use. With Data Shifting Insert Mode, you can enter data and have SPFLite maintain the alignment of columns in a way similar to how a word processor would maintain the alignment of tabbed columns (without actually using tabs).

When Data Shift Insert Mode is in effect:

- non-blanks are shifted right as new data keys (including spacebar) are pressed, as usual
- when existing non-blanks are pushed to the right, and a span of two or more blanks follows the non-blanks, the span of blanks is shortened
- a span of blanks will never be completely removed by shortening it; there will always remain at least one blank
- the status indicator will show **INS** instead of INS or OVR
- pressing either of the the keys mapped to [\(Insert\)](#) or [\(DataInsert\)](#) will cause the status indicator to revert from **INS** back to OVR

For example, assuming the following data lines:

000001	A, B	CCC
000002	AA, BB	CCC
000003	AAA, BBB	CCC

Suppose we want to add 1, 22 and 333 before the commas on the data lines above. Using regular Insert Mode, this would first produce:

000001	A1, B	CCC
000002	AA22, BB	CCC
000003	AAA333, BBB	CCC

and the lines would have to be 'repaired' to restore the alignment of the CCC column. With Data Shifting Insert Mode, the strings ending in B would be pushed to the right without moving the CCC column, as long as no data loss occurred. On line 3, that is not possible, and so its CCC value has to be pushed over:

```
000001 A1,B      CCC
000002 AA22,BB    CCC
000003 AAA333,BBB CCC
```

When multiple spans of blanks exist, they get "compressed" in left-to-right order, starting from the point on the line where the cursor is positioned and going rightward as needed.

When a key mapped to [\(DataInsert\)](#) is pressed, the editor enters Data Shifting Insert Mode. To distinguish this mode from regular Insert Mode on the Status Line, the regular INS indicator is displayed in Green hi-lite as **INS** instead.

When the editor is in Data Shift Insert Mode, another function key mapped to either [\(Insert\)](#) or [\(DataInsert\)](#) will return the status line to the OVR indicator.

Data-Delete function

This function will delete text in the same way that [\(Delete\)](#) does, except that the data being "pulled left" by the delete action is "delimited" by any span of two or more blank characters. Because of this, if you are deleting data that is formatted into columns, [\(DataDelete\)](#) will not disturb the column alignment, as long as two or more blanks separate the columns.

Unlike the [\(DataInsert\)](#) function, [\(DataDelete\)](#) does not define or toggle a "mode" but merely deletes a character, as described below.

A suggested mapping for this key is Ctrl-Delete or Shift-Ctrl-Delete.

Let's take the same data from above, inserting an additional line, and compare [\(Delete\)](#) and [\(DataDelete\)](#). Assuming the following data lines:

```
000001 A,B      CCC
000002 AA,BB    CCC
000003 AAA,BBB  CCC
000004 AAAA,BBBB CCC
```

Suppose we want to delete the words A, AA, AAA and AAAA. Using regular [\(Delete\)](#) function, this would produce:

```
000001 B      CCC
000002 BB     CCC
000003 BBB    CCC
000004 BBBB   CCC
```

and the CCC column gets shifted over. Using the [\(DataDelete\)](#) function produces:

```
000001 B      CCC
000002 BB     CCC
000003 BBB    CCC
000004 BBBB   CCC
```

Note that the CCC column gets shifted over on line 4. That is because there was only one space between the original BBBB and the CCC, and so the string "BBBB CCC" is treated as a "single unit" of text, and is pulled to the left. On lines 1-3, the CCC string in each case was preceded by two or more spaces, and so it does not get pulled to the left on those lines.

Data-Backspace and Data-Delete-Mark

Two new functions are available that involve data shifting. These are [\(DataBackspace\)](#) and [\(DataDeleteMark\)](#).

[\(DataBackspace\)](#) is essentially a cursor-left-one followed by a [\(DataDelete\)](#).

[\(DataDeleteMark\)](#) will delete highlighted (marked) text in the same way that [\(DataDelete\)](#) does, but if there is no text highlighted, the function does nothing.

Defining Tabs and Column Markers

Contents of Article

[Introduction](#)

[Tabs](#)

[Column markers](#)

[Defining tabs and column markers](#)

[Treatment of undefined codes on TABS and MARK lines](#)

[Affect of tab positions on the COLS line](#)

[Using the + code to define virtual tabs](#)

[Using the < and > codes for column markers](#)

Introduction

Tabs and column markers are aids to help you position text horizontally when data position is important or desirable. This may be for language syntax indenting, for standard positioning of comments, or for any other reason where fixed column positions for data is desirable.

Tabs

Tabs are markers indicating the horizontal positions to which the TAB key will successively position the cursor on a line. This enables you to quickly and accurately move the cursor to predefined column locations.

Column markers

Column markers are faint vertical lines visible on the screen at specified columns. They have no effect on Tabs or the use of the TAB key; the two are independent of each other. They are intended to provide visual guidance only. However, a tab column is typically aligned at a column marker because it's usually convenient to combine them that way. Most users having many tab stops will only have a few column marker lines defined, but they probably would place those column markers where tab columns are defined.

Defining tabs and column markers

The methods for defining these two items are similar.

To define Tabs or Column Markers, first obtain a visible copy of the definition line as follows:

Type **TABS** or **MARK** on any Line command area, a Definition line will be inserted.

```
000010 Data text here
=TABS>      *      *
000011 More Data text here
```

or

```
000010 Data text here
=MARK>      *      *
000011 More Data text here
```

will appear.

Enter an * asterisk character at each position you wish to be a Tab Stop or Column Marker. To ensure you have the correct column, you may wish to insert a **COLS** line for reference, or the Status line displays the column number.

Note that SPFLite will remember the Tabs and Mark lines and associate them with the type of file being edited. The next time this file type is edited, the Tabs and Mark lines will be re-established when the file is loaded into the editor. This will enable you to have different 'favorite' tabs definitions for each type of file you edit.

The editor can also be told to temporarily ignore the tabs line without altering its definition by issuing the primary command **TABS OFF**; normal tab operation can be resumed later with a **TABS ON** command.

Similarly, the **MARK OFF** Primary command can be used to temporarily suppress the display of the MARK lines without actually modifying the MARK line itself. Display of the MARK lines can be resumed with a **MARK ON** command.

The color used to draw the Column Marker line may be specified in the Options - Screen settings.

Treatment of undefined codes on TABS and MARK lines

The TABS line uses code * and + to define tab positions, and the MARK line uses * < and > to define column markers. Any nonblank characters on these lines, other than the defined codes, are ignored and treated as comments.

Affect of tab positions on the COLS line

When TAB positions are set, either by the * code for standard tab positions or by the + code for virtual tab positions, the COLS line will show underscores for every effective tab location, such as this:

```
=COLS> -----+---1---+---2---+---3---+---4---
```

Using the + code to define virtual tabs

If you need to define tabs for repetitive column positions, you do not have to manually enter every single tab stop. For example, if you wanted tab stops in columns 1 and 10, and every 5 columns starting in column 16, entering the repetitive stops for a file with very long lines would be a lot of work.

```
=COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+
=TABS> * * * * * * * * * * * * * *
```

SPFLite supports defining *virtual tab stops* using a + code instead of an * code. Virtual tab stops work as follows:

- The + code must be the last code on the =TABS> line
- There may be at most one + code
- The + code must be preceded on the left by at least one * code

When a =TABS> line is defined in this way, the distance in columns between the + code and the last * code defines a *virtual tab step*, which is the number of columns between each subsequent virtual tab stop.

Let us define the tabs above using the virtual tab code +. The + code will code in column 21, and the * code in column 1, 10 and 16. Because the distance between column 21 (where the + code is located) and column 16 (where the last * code is located) is 5, the virtual tab step is 5, and so the repeating virtual tab stops will appear every 5 columns, starting at column 21. The =TABS> line now looks like this:

```
=COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+
=TABS> * * * +
```

Now, no matter how far to the right you may scroll, there will always be a tab stop at every 5 columns, whether that be at column 101 or 1006.

Using the < and > codes for column markers

SPFLite also supports two alternate codes of < and >.

When a < code is used on the MARK line, the faint vertical line appears to the left of the < sign, like this: <|
When a > code is used on the MARK line, the faint vertical line appears to the right of the > sign, like this: >|

It may be seen that the `<` and `>` codes “point” to the side of the column where the column marker line is to appear, and thus are easy to remember.

It is legal to have `>` and `<` next to each other as in `><` on the same MARK line, and you will see this: `><`

The * code from prior versions of SPFLite remains available, and works the same as the < code.

Here is a sample of how these lines actually appear. Note that the MARK lines were made somewhat dark on this sample screen to be sure they were visible in this Help document. In practice, it is helpful to make these very faint so you can just barely see them; that way they can act as a guide but not be a distraction. The exact color for the MARK lines is set in the [Screen tab of Global Options](#) under Column Marker Line. As can be seen, the Mark lines appear on actual data text lines, not on special lines such as the =TABS> and =MARK> lines themselves.

Windows Clipboard, Cut and Paste

Contents of Article

[Introduction](#)
[Classic ISPF-style CUT and PASTE](#)
[SPFLite clipboard mode](#)
[Windows-Style cut and paste](#)
[SPFLite Named Private Clipboards](#)
[Copy and Cut operations](#)
[Paste operations](#)
[Hybrid Copy primitive \(CopyPaste\)](#)
[Power Typing and the clipboard](#)

Introduction

SPFLite provides two basic methods to support Cut and Paste and the Windows clipboard.

Clipboard support is only for plain-text data. While Windows allows many types of data formats to be stored in the clipboard, only those formats which can be converted to plain text (that is, ANSI characters) can be used.

Technically, Windows categorizes these formats as TEXT, OEMTEXT and UNICODE. The standard ANSI (Windows 1252) character set, which SPFLite uses internally for all its data, is what TEXT is.

In practice, if you can paste characters into the Windows Notepad editor, and they display correctly, you can probably use them in SPFLite. If you try to paste text from a highly-formatted document in Microsoft Word, for example, you will lose all special formatting and some data may be lost or appear differently. And, if you tried to paste some highly specialized document, like a CAD drawing, you will probably get nothing.

Classic ISPF-style CUT and PASTE

The first supported method operates like the original ISPF **CUT** and **PASTE** commands. The operations work on whole lines only, not on partial line contents.

CUT command

Selection of data for Cut operation is done via normal use of [C/CC](#) or [M/MM](#) line commands. Following selection of the data, a [CUT](#) command is entered on the command line. If [C/CC](#) was used to select the data, the lines are **copied** to the clipboard; if [M/MM](#) was used, the lines will be copied to the Clipboard and then **deleted**.

Note: The **CUT** primary command will do both **cutting** and **copying** actions. The command name **CUT** adheres to the IBM ISPF naming convention for this command, even if it is a bit "inaccurate" of a term. IBM decided on the command name **CUT** for ISPF a long time ago, before "cut" and "copy" had the more precise, commonly understood meanings they have today. If the line range is defined by a **C/CC** block, or by any type of line-control-range operand on the primary command line, a **copy** operation takes place. If the line range is defined by an **M/MM** block, a **cut** operation takes place.

Bear in mind that the **COPY** primary command is used for copying outside files into the current edit session, and has nothing to do with the clipboard. To copy from the clipboard, the **PASTE** command is required.

PASTE command

To paste lines from the Clipboard, enter an [A](#), [B](#), [O/OO](#), [OR/ORR](#) or [H/HH](#) line command to indicate **where** the lines are to be inserted, and then enter [PASTE](#) on the command line. If [A](#) or [B](#) was used to indicate position, the Clipboard lines will be inserted at the indicated point. If [O/OO](#) was used the Clipboard lines will

be overlaid on the indicated lines exactly like a Copy / Overlay operation using [C/CC](#) and [O/OO](#) line commands. If the [H/HH](#) line commands were entered, the lines marked with [H/HH](#) will be **replaced** by the pasted lines.

Note: if the file is empty at the time [PASTE](#) is entered, no [A](#) or [B](#) line command is needed. The Clipboard lines will simply be loaded into the file text area. Thus, a [PASTE](#) into an empty is treated like there was an [A](#) line command on the Top of Data line.

SPFLite clipboard mode

SPFLite supports a startup mode known as **CLIP**. This mode is specified by coding the characters '**-CLIP**' as a command-line operand when invoking SPFLite. In this mode, SPFLite will automatically load the contents of the Windows clipboard when started, and will save the current edit data back to the Windows clipboard on termination. See a discussion of the **-CLIP** command-line option under [Starting SPFLite](#). In addition, the Primary command [CLIP](#) may be entered at any time to open a new edit tab containing the contents of the current Clipboard. After you are finished, the contents are returned to the clipboard when the tab is closed.

Note: It is important to understand that a CLIP Edit session **copies** the clipboard into an SPFLite-controlled edit session dedicated to clipboard editing, and then **copies it back** to the actual clipboard when you're done.

Remember: The clipboard, and the clipboard edit session, are **not** the same thing.

While you are **in** the CLIP Edit session, you are not actually modifying the Windows clipboard - only a copy of what it was, prior to the CLIP Edit session beginning. What that means is that, while you are in a CLIP Edit session, you could jump outside of it, to another SPFLite edit tab, or outside of SPFLite itself, then copy **more** data to the Windows clipboard, and paste **that** into your CLIP Edit session (or into **another** CLIP Edit session), and you could continue doing that process as many times as you like. Only once you're done will the edit session be copied back to the Windows clipboard. This fact can make for some very interesting and powerful editing techniques.

Note: It is possible to have **many** CLIP edit sessions active at the same time. Each CLIP Edit session will be initialized with the current contents of the Windows clipboard **at the time it is initiated**. You might want to do this to create some "temporary editing sessions". This is a perfectly legal (if a bit unusual) thing to do, but if you decide to do this, remember that each time you close a CLIP Edit session, the contents of **that** session are copied back to the real Windows clipboard. Thus, if the final contents of the Windows clipboard when you are all done is important to you, you may have to carefully choose **which** CLIP Edit session you close last.

Windows-Style cut and paste

To support Windows style cut and paste, SPFLite has several primitive keyboard functions. By *primitive*, we mean functions like TAB, NewLine, Insert, Delete etc. These functions take effect immediately when used, and are neither a Primary or Line command, and must be mapped to a key in order to be used.

These functions can be seen when customizing the keyboard (see "[Keyboard Customization and Keyboard Macros](#)") and the functions may be assigned to whatever mappable keys you wish. The descriptions below are written in terms of the normal default assignment for these new keys.

SPFLite Named Private Clipboards

SPFLite supports a feature called Named Private Clipboards.

Named Private Clipboards are areas where the current contents of the edit file may be saved and restored. A Named Private Clipboard may be directly created or pasted into an edit file without going through the standard Windows clipboard first.

A Named Private Clipboard is a permanent area, stored as a file having a file extension of **CLIP** and stored in the SPFLite directory under a folder called **CLIP**. SPFLite takes the "name" of the named clipboard and forms a file name of "**name.CLIP**".

Named Private Clipboards are available and shared with all active edit tabs, and are saved by SPFLite between sessions. So, whenever you start SPFLite, the named private clipboard contents from your last session are always available for use. You could, for example, use these to store frequently used 'boilerplate' text.

Use of Named Private Clipboards is entirely optional, and in no way affects the normal operation of the Windows clipboard functions in SPFLite.

A named clipboard is referred to by its name, which may be any user-selected name (which, as a "name", should avoid special characters and blanks). The name of a Named Private Clipboard should not be the same as any SPFLite keyword. In particular, you would run into problems if your Named Private Clipboard had any of the names **BEFORE** **AFTER** **ERASE** **REP** **REPL** **APPEND** **X** or **NX**.

To cut text into a Named Private Clipboard, you simply include the *name* in the **CUT** primary command. To paste text from a Named Private Clipboard, you include the *name* in the **PASTE** primary command. See [CUT](#) and [PASTE](#) for more information.

You can edit a named clipboard just as easily as editing the Windows clipboard, by specifying **CLIP** *name*, where *name* corresponds to a named clipboard created by a **CUT** *name* command.

Keyboard functions that involve the clipboard can take an option to specify a Named Private Clipboard. The format of this option is a / slash followed by the name of the Named Private Clipboard. For example, suppose a Named Private Clipboard of **myclip** exists. To paste from the Windows clipboard, you would use the function [\(Paste\)](#), but to paste from the **myclip** private clipboard, you would use [\(Paste / myclip\)](#). This technique works for every keyboard function that uses a clipboard, such as [\(Copy\)](#), [\(CopyPaste\)](#), [\(Paste\)](#), [\(ClipPath\)](#), etc.

Copy and Cut operations

In order to perform a Copy or Cut, you need to select the desired text. Text may be selected by dragging over the text area with the left mouse button held down, or by use of the Arrow keys with the Shift key held down. The area being selected by these operations will be displayed in inverse video. (Note that using the Shift + Arrow keys is a default key mapping to the MARK functions. You can reassign these functions to any desired keys.)

To select a "2D" multi-line block of text with the mouse, you must hold down a Shift, Alt or Ctrl key first, unless you have enabled the Global Options checkbox that allows multi-line selects without the extra key. See [Allow 2-D mouse selection without Shift/Ctrl/Alt](#) for more information.

Keyboard selection may proceed past screen boundaries if keyboard scrolling is activated ([Options - Keyboard](#)), but mouse selection is always restricted to a single visible screen. Mouse selection now supports a double-click to select a 'word' and highlight it.

The keyboard selection uses the keyboard functions [\(MarkLeft\)](#), [\(MarkRight\)](#), [\(MarkUp\)](#), [\(MarkDown\)](#) and [\(MarkEnd\)](#). These primitive functions are mapped to the Shift-Left, Shift-Right, Shift-Up, Shift-Down and Shift-End keys, which are commonly used in other text editor programs.

Note that using the Shift + Arrow keys is merely a default key mapping to the MARK functions at installation time. You can reassign these functions to any desired keys.

You may also highlight a large span of lines using the [T/TT](#) line command.

Once selected, the data can be Copied with the [\(Copy\)](#) function, Cut with the [\(Cut\)](#) function or Lifted with the [\(Lift\)](#) function. These primitive functions are mapped by default to the Ctrl-C and Ctrl-X keys, just as in Windows.

Paste operations

The [\(Paste\)](#) function will paste the contents of the Windows clipboard, whether a single line or multiple lines.

Pasting may be done into any text line or the primary command line field. It is **not** a full line paste like Classic ISPF paste; you may paste the text at any point in the existing data in the line. The paste command is also **sensitive** to the current INS/OVR mode. If you are in Insert mode, the paste text will be inserted and existing text shifted right. If you are in OVR (non-insert) mode, the paste data will simply overtype the existing characters in the line.

The mapping of the primitive paste function ([\(Paste\)](#)) defaults to the Ctrl-V key, just as in Windows.

Hybrid Copy primitive (CopyPaste)

There is a hybrid primitive function called ([\(CopyPaste\)](#)) which can be used to make a dual-purpose key. When the key is pressed, if there is text selected on the screen, it acts exactly like the ([\(Copy\)](#)) function. If there is **no** selected text, it acts like the ([\(Paste\)](#)) key. A typical use of this would be to assign ([\(CopyPaste\)](#)) to the right mouse button (see [Options-Mouse](#) for how to do this).

This provides a powerful editing action when doing cut/paste, and operates like the QuickEdit facility on the Windows command line.

Power Typing and the clipboard

When you are in Power Typing mode, you can use the Windows-style clipboard functions described above. In addition, you can use the functions that are specific to Power Typing. These are ([\(PowerCopy\)](#)), ([\(PowerCut\)](#)) and ([\(PowerPaste\)](#)). See the section on [Working with Power Typing Mode](#) for more information.

Saving the Edit STATE

Contents of Article

- [*Introduction*](#)
- [*STATE and the File Type*](#)
- [*STATE Integrity*](#)
- [*STATE Command Options*](#)
- [*Displaying the STATE setting in the Profile*](#)
- [*Where is the STATE saved ?*](#)

Introduction

When working on a file, particularly files such as source files, a fair amount of effort may be expended on establishing line-labels to act as bookmarks, setting line tags to mark particular lines and excluding various ranges of lines so that only the current pertinent lines are visible and being worked on. As well, you may have added [**NOTE**](#) lines to the file.

Your work on a large source program might continue over an extended period of many days. But every time you close the editor, all that effort noted above would normally be lost, and has to be re-established the next time the file is loaded for edit.

SPFLite can now save all the above as **STATE** information and automatically re-establish the edit session when the file is loaded back to its status when closed. Line labels, line tags, **NOTE** lines, excluded line ranges and current top of screen location are all retained on close and recreated when the file is opened. The file is redisplayed exactly as you left it at close time.

This action is often called "reestablishing the edit context". By doing this, it eliminates the work of you having to do all of that manually, and will greatly aid your productivity. You'll spend less time going, "where was I, what was I doing?" and you'll be able to resume your work much more quickly.

STATE and the File Type

So for source files, STATE seems a nice feature, and since STATE is a setting maintained in the file type Profile, you can have STATE set ON for source file types where it is appropriate, and OFF for file types where it would provide no benefit.

You can now also have selective STATE processing within a single file type. Continue reading and pay attention to STATE Setting Options below.

STATE Integrity

In order to make sure the saved STATE information is properly synchronized with the file it's associated with, there are calculated hash values kept in the STATE file. If the validation of these values fails when the file is opened, the STATE information will not be used, and you will receive an error message.

This can happen if you use an external program to modify the file, or another editor to insert, delete or move lines around in the file. For example, if you created a label of **.ABC** on line 10 of a file, and then modify the file with an outside editor, the label **.ABC** can no longer be reliably used to reference line 10, because the underlying data on that line is (probably) not the same any more.

An invalid STATE condition can happen if you restore or copy a file from another location, such as a backup, or download a copy from a web site. When SPFLite detects this, it will not use any saved STATE information.

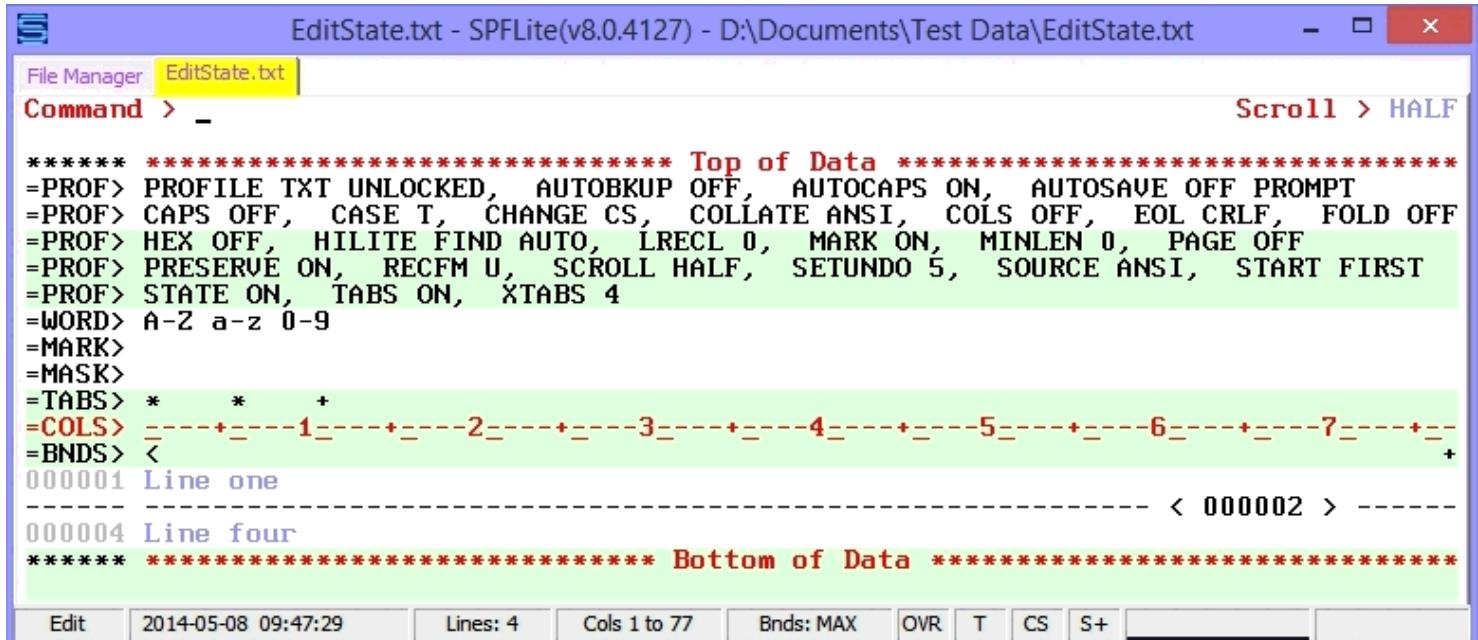
STATE Command Options

STATE now supports multiple operand options to allow you to customize how STATE processing is handled. These operands are:

STATE ON	This indicates that you would like all files controlled by this Profile to be processed by STATE.
STATE OFF	This indicates that no files controlled by this Profile are to be handled by STATE.
STATE MOST	This is similar to STATE ON. If you take no specific action for a specific file, then it acts just like STATE ON, all files will be processed by STATE. However it supports allowing you can exempt specific files, see STATE DELETE below.
STATE FEW	This is similar to STATE OFF. If you take no specific action for a specific file, then it acts just like STATE OFF, all files will not be processed by STATE. However you can exempt specific files, see STATE CREATE below.
STATE DELETE	This command would be issued while editing a specific file and STATE MOST has been set. In fact, it will be rejected if these conditions are not met. Since STATE MOST, by default, will have created active STATE data for this file, the STATE DELETE will remove this active data and setup an indicator so that no future STATE processing for this file will occur.
STATE CREATE	This command would be issued while editing a specific file and STATE FEW has been set. In fact, it will be rejected if these conditions are not met. Since STATE FEW, by default, will not have created any active STATE data for this file, the STATE CREATE will cause this active data to be created the next time the file is saved, and will setup an indicator so that STATE processing will continue for this file in future edits.

Displaying the STATE setting in the Profile

Whether or not your editing context is saved for you or not is controlled by a Profile setting called [STATE](#). If you issue a **PROFILE** (or just **PRO**) command, here is what the display might look like:



The screenshot shows the SPFLite editor interface with the file 'EditState.txt' open. The 'Command' tab is selected. The text area displays the following profile settings:

```
***** ***** Top of Data *****
=PROF> PROFILE TXT UNLOCKED, AUTOBKUP OFF, AUTOCAPS ON, AUTOSAVE OFF PROMPT
=PROF> CAPS OFF, CASE T, CHANGE CS, COLLATE ANSI, COLS OFF, EOL CRLF, FOLD OFF
=PROF> HEX OFF, HILITE FIND AUTO, LRECL 0, MARK ON, MINLEN 0, PAGE OFF
=PROF> PRESERVE ON, RECFM U, SCROLL HALF, SETUNDO 5, SOURCE ANSI, START FIRST
=PROF> STATE ON, TABS ON, XTABS 4
=WORD> A-Z a-z 0-9
=MARK>
=MASK>
=TABS> * * +
=COLS> -----1-----2-----3-----4-----5-----6-----7-----+
=BNDS> <
000001 Line one
----- < 000002 > -----
000004 Line four
***** ***** Bottom of Data *****
```

The text area is highlighted with green and red colors to indicate different sections of the profile settings. The status bar at the bottom shows the date and time (2014-05-08 09:47:29), line count (4), column range (1 to 77), and various mode indicators (BndS: MAX, OVR, T, CS, S+).

Notice the =PROF> line that says **STATE ON**. In this example, this means that for files of type TXT, state information will be saved. Note also that **S+** appears in the Status Bar indicating that an active STATE file exists for this file.

When **STATE** is **ON**, the state of your [excluded lines](#), [User Lines](#), [labels](#) and [tags](#) will be saved when you close your file, and restored when you reopen it.

[NOTE and xNOTE lines](#) require **STATE ON** to be in effect if you wish these lines to be retained between edit sessions.

Some (but but not all) options of [START](#) (which relies on a persistent label of .START) also require **STATE ON**.

Where is the STATE saved ?

The state information for a given file is saved in a separate file within the STATE folder in the SPFLite data directory. Since STATE data is normally related to your current work with a file, STATE files which have not been updated for 180 days will be automatically purged from the STATE folder. This will prevent the accumulation of obsolete STATE files which are no longer of any use.

Working with Tab Pages

Contents of Article

- [*Introduction*](#)
- [*Data isolation*](#)
- [*Switching between tabs*](#)
- [*Opening a new tab*](#)
- [*Closing a tab*](#)
- [*Indicating file-modified status in the tab header*](#)
- [*Example*](#)

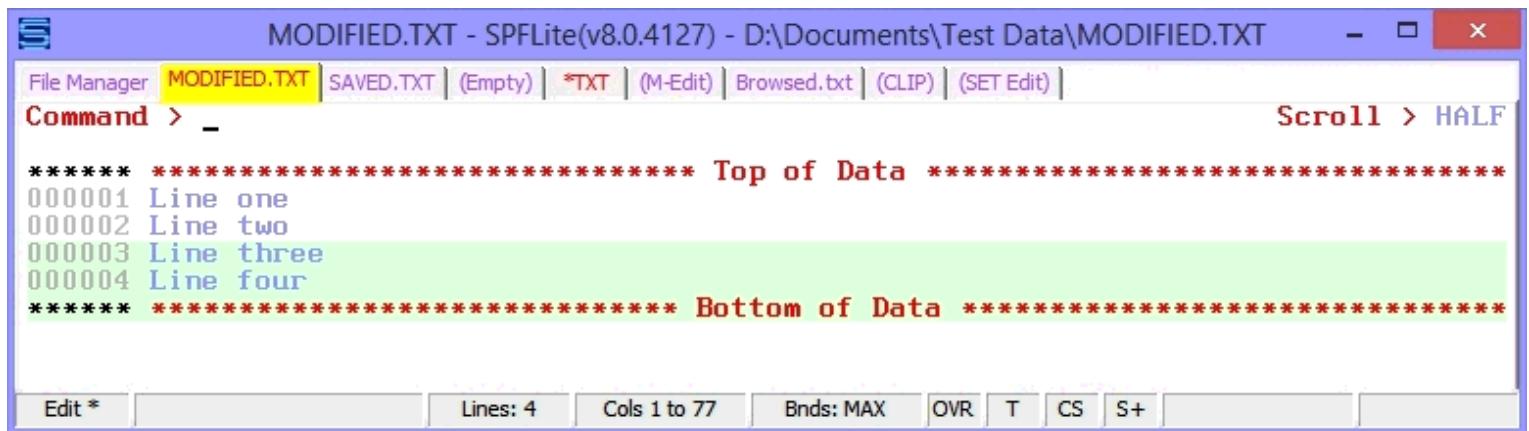
Introduction

SPFLite divides the screen into multiple **tabbed sections**, to support the editing or browsing of multiple files from a single SPFLite instance.

Each open file is displayed and manipulated within a separate tab page in the SPFLite window. These sections are often referred to as "file tabs" or just "tabs", but may contain any of the following within a tab:

- **File Manager.** Is always the left-most tab. The File Manager may display a directory list, or a File List.
- **Edit tab.** Contains a file opened for editing.
- **Empty tab.** Contains a new, empty file that has no name and has not yet been saved.
- **Cloned tab.** Contains a copy of an existing file, which has not yet been given a new name and has not yet been saved. A temporary name with an * and a file extension will appear.
- **Multi-Edit tab.** Contains a set of multiple files opened for editing.
- **Browse tab.** Contains a file opened for browse. Browse is a read-only edit session, in which changes cannot be saved.
- **Clipboard tab.** Contains the contents of the Windows clipboard, allowing you to directly edit it. At most one clipboard tab may be present.
- **SET Edit tab.** Contains the values for all defined SET variables. At most one SET Edit tab may be present.

The screen below shows SPFLite with nine open tabs.



The base file name of a file being edited (without the file's path) is used as the label for each tab. When a file tab is selected (made active) the tab itself is displayed with a different background color, and the full filename, including the path, is shown in the SPFLite windows title bar.

Data isolation

The data being edited in each tab is completely separate from the data in any other tabs. However, some SPFLite program data is used in common with all tabs. This includes:

- The global settings controlled by the [Options](#) settings
 - The Command retrieve stack used by the [RETRIEVE](#) command
 - The Windows Clipboard
 - Any Named Private Clipboards you may have saved
 - The value of any **SET** symbol names
 - The contents of Globally Stored Values saved by programmable macros

Switching between tabs

To switch between tabs you can either use the mouse to left-click on the desired tab, or you can use the SPFLite primary commands [SWAP PREV](#) to switch to the previous tab, and [SWAP NEXT](#) to switch to the next tab. Note: Both **SWAP PREV** and **SWAP NEXT** will wrap if there are no more tabs in the indicated direction.

There is also a **SWAP PRIOR** command which can be used to alternate displays of two tabs. **SWAP PRIOR** swaps to the previously displayed tab. **Note:** If you click on one tab, and then click on a second (different) tab, then from that point on, each time you issue a **SWAP PRIOR** command, it will alternate the active screen between those two tabs. That is, **SWAP PRIOR** will always alternative between the last two active file tabs.

If you use these functions in other software, you may wish to map **SWAP NEXT** to Ctrl-Tab and **SWAP PREV** to Ctrl-Shift-Tab for compatibility with these applications.

Whether you switch among file tabs using the mouse or the **SWAP** commands, SPFLite will maintain your current cursor position in every file tab, where you will find it if you leave a file tab and then return back to it.

Opening a new tab

When SPFLite is started, the File Manager tab will be displayed with your default directory list displayed. If you have enabled the Global Option for [Re-Open last file\(s\) at Start](#), the files that were left open the last time SPFLite was closed will also appear in their own tabs.

To open a file for Edit or Browse, enter the appropriate command character (**E** or **B**) next to the desired files. Each selected file will be opened in its own tab. You can also select multiple files to be combined in a single Multi-Edit session with the **M** line command. See [Working with Multi-Edit Sessions](#) for more information.

In File Manager, Left-Click on **New** in the Quick Launch bar (or enter a non-blank to its left), or enter the **EDIT NEW** command, to open a new, empty file.

You can return to the File Manager tab at any time to select other additional files for processing, by clicking on the File Manager tab, or by issuing a **SWAP HOME** command.

Closing a tab

A tab will be closed when an [END](#) command is issued. [END](#) may also save the data depending on your [AUTOSAVE](#) setting. Following END processing the tab will be closed and disappear from the screen.

If you issue a **CANCEL** command rather than **END**, the edit data is deleted and the tab returned to an (Empty) status. The tab is **not** closed by the **CANCEL** command.

You may also close a tab by Right-clicking on the tab header. This is treated as equivalent to issuing an **END** command and all normal END processing takes place as usual.

If you click on the main window close button **X**, SPFLite will close every tab. As each file is closed, the file is either saved or not saved, with or without a prompt to you, depending on how the [AUTOSAVE](#) option is set in the Profile for that file's type.

Indicating file-modified status in the tab header

SPFLite can provide a visible indication when a file has been modified but not yet saved, by altering the color of the text in the Tab header for that file.

The colors to be used are under your control, and are specified in the [Options - Screen](#) dialog.

In order to benefit from this feature, it's up to you to pick good, contrasting colors. SPFLite does not predefine the color palettes during installation to accomplish this. The dialog below shows an example of how this might be done:

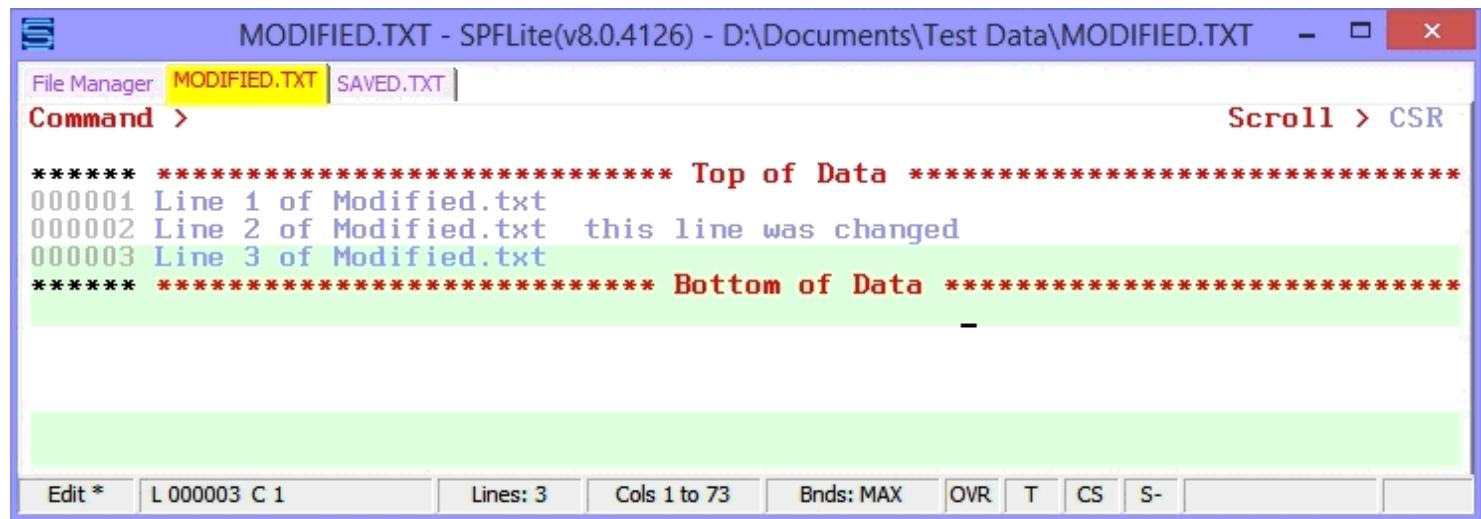
General | FM | Submit | **Screen** | KBD | Status | Schemes | HiLites

<input checked="" type="checkbox"/> Vertical cursor in Ins. Mode.	<input type="checkbox"/> Horizontal cursor ruler?		
24 % Height Normal Cursor.	<input type="checkbox"/> Vertical cursor ruler?		
100 % Height Insert Cursor.	<input checked="" type="checkbox"/> Alternating Background screen colors?		
DejaVu Sans Mono Bold	Font Name	<input checked="" type="checkbox"/> Hilight Command line keywords?	
12	Font Pitch	<input type="checkbox"/> # K-Board help lines to show?	
<input type="button" value="Choose"/>	(or Choose Font here)	<input type="checkbox"/> Width of Line Numbers (5-8)	
<input type="checkbox"/> Column Marker Line Color	FG BG1 BG2		
Line Numbers High Intensity	<input type="color"/>	<input type="color"/>	<input type="color"/>
Line Numbers Low Intensity	<input type="color"/>	<input type="color"/>	<input type="color"/>
Active Tab Modified	<input type="color"/>	<input type="color"/>	<input type="color"/>
Active Tab Unmodified	<input type="color"/>	<input type="color"/>	<input type="color"/>
Inactive Tab Modified	<input type="color"/>	<input type="color"/>	<input type="color"/>
Inactive Tab Unmodified	<input type="color"/>	<input type="color"/>	<input type="color"/>
PFK Help Lines	<input type="color"/>	<input type="color"/>	<input type="color"/>
Status Line	<input type="color"/>	<input type="color"/>	<input type="color"/>
FM Tool Bar	<input type="color"/>	<input type="color"/>	<input type="color"/>
Error Messages	<input type="color"/>	<input type="color"/>	<input type="color"/>

 INI File is: D:\Documents\SPFLite\SPFLite.INI

Example

Here is an example of how these color choices would appear in the main edit window. The current file tab shows **red** lettering because **MODIFIED.TXT** is **modified**, while **SAVED.TXT** is in **blue** lettering because it is **unmodified**.



Working with Command Chaining

Introduction

SPFLite provides a new editing concept: **Command Chaining**. To help you understand this concept as quickly as possible, the key features of Command Chaining are presented below in question and answer format.

What is Command Chaining ?

Command Chaining allows you to perform a series of Edit primary commands, passing a common set of data lines from one command to the next, so you can apply several command operations to the same set of lines.

Only Primary commands can be chained. There is presently no support to chain **line** commands. This facility could properly be called the "Primary Command Chaining" facility, but we'll drop the "Primary" to be brief.

Command Chaining only applies to data lines, not to "special" lines like NOTE lines.

What kind of primary commands can be chained ?

Chaining of primary commands is provided to perform a set of operations on a related set of data lines. Such commands are called **chainable**.

Chainable commands can be found in the Primary Commands section of the Help document. Only commands that show a **syntax** containing a **line-control-range** operand are chainable.

Commands like **UP** and **DOWN** do not take a line-control-range, so they are not chainable. However, "non-chainable" commands have no impact on commands that **are** chainable. So, if a non-chainable command like **UP** or **DOWN** is issued in between chainable commands, it does not affect or disturb the way the chain operates. In this sense, non-chainable commands are "ignored" for purposes of the command chain.

Is Command Chaining related to issuing multiple commands using a command separator ?

No, not directly. The command separator character, normally the ; semicolon, allows you to issue more than one primary command on the same Edit primary command line. For example,

```
X ALL ; FIND ABC ALL
```

first excludes all lines in the file, and then finds (and unexcludes) all lines containing the string **ABC**. However, the **X** and **FIND** commands have no relationship to each other. They are simply two commands that happen to be performed one after the other. **These two commands here are not a command chain.**

You may find issuing a command chain using multiple commands separated by semicolon to be convenient, but that is not required. Chained commands can also be issued individually, without the semicolon notation.

Why would I want to use Command Chaining ?

The main reason to do this is when you want to apply a number of commands to a set of lines that are determined by the outcome of some initial command like **FIND ALL** or **CHANGE ALL**.

Couldn't I use regular tags, X|NX lines, or U|NU lines to do what Command Chaining does ?

Yes, and for some situations, you may still want to do that. However, those techniques require a "set-up phase" where you have to define a set of tagged or marked lines. That is useful if your set of lines is "going to be around for a while".

The advantage of Command Chaining is that when you have a common set of lines needed by several commands, but it's a "one time" situation, using a Command Chain is easier, because you don't need any "set-up" process. You can simply chain your commands together, without needing to first define tags or User Lines, and then having to remove such tags or User-Line marks afterwards.

How is Command Chaining different than multiple commands per command line ?

SPFLite maintains "lists" of which lines are "involved" in a particular command. You can then use one of these "lists" in a subsequent command. The use of a such a list is how command chaining is implemented.

The multiple commands involved in a sequence of chained commands can appear on one primary command line (separated by semicolons), or on separate lines. So, the two features can be combined or used individually, since they are separate and distinct concepts.

What do I need to know about these "lists" of lines ?

For Command Chaining, SPFLite deals with four kinds of "lists" of data lines when executing a primary command. These are **eligible** lines, **reached** lines, **affected** lines and **unaffected** lines.

Each list of lines may contain zero or more data lines, and are discussed below.

What are **eligible** lines ?

When a primary command is issued that accepts a line-control-range operand, SPFLite takes into account any **X|NX**, **U|NU**, line-range labels and/or tags that may be specified, and whether **PREV**, **NEXT**, **FIRST**, **LAST** or **ALL** is used, as well as any implied operands.

The list of all lines that SPFLite could possibly look at, in order to perform a command such as **CHANGE**, is the **eligible list**.

Unless a data line is eligible, it will not be involved in command chaining.

What are **reached** lines ?

A reached line is an eligible line that a primary command has "looked at" in the course of its operation. The list of all reached lines is the **reached list**.

Why would the **reached lines** be different than the **eligible lines**?

The main reason is whether the operand **ALL** appeared on the command, or if **ALL** was implied by a particular command.

For example, suppose you had a 10 line file, and only line 5 contains the string **ABC**.

If the file is positioned at the Top of Data, and you said **CHANGE ABC DEF**, the command has an implied **NEXT** operand. That is, it will change the next **ABC** into **DEF**, if one exists. In your file, the **CHANGE** command would start at line 1 looking for **ABC**. When it got to line 5, it finds **ABC**, changes it to **DEF**, and stops. So, it has **reached** lines 1 through 5, because it had to "look" at those lines in order to find the first **ABC** (the "next" one, starting from line 1). Since the **CHANGE** command stopped looking after it got to line 5, **lines 6 through 10 were not reached**.

However, if you said, **CHANGE ABC DEF ALL**, SPFLite has to "look" at every line in the file, to be certain that every occurrence of **ABC** is located. Thus, it will have **reached** all lines in the file, from 1 to 10. In that case, the command will have reached every eligible line, and so the eligible list and the reached list would be the same set of lines.

How can I remember the difference between **eligible** and **reached** lines ?

When a primary command has the **ALL** operand, the eligible and reached lines will be the same.

When certain primary commands like **UC** are used, they are applied to all lines in the line-control-range, even if you don't use the **ALL** keyword. For **UC**, the "**ALL**" is implied. In such commands, the eligible and reached lines will be the same.

When the primary command does not have either an explicit **ALL** or an implied **ALL**, it is possible the number lines reached is fewer than the number of eligible lines.

What are **affected** lines ?

An **affected** line is a *reached* line on which the primary command does something, such as find or change a string, or modify the data on the line in some way. The list of all affected lines is the **affected list**.

What are **unaffected** lines ?

An unaffected line is a *reached* line on which the primary command did not find a search string or otherwise did not modify the data on the line in any way. The list of all unaffected lines is the **unaffected list**.

To be precise, an unaffected line is a reached line that does not fall under the category of "affected". That is, you could say, **unaffected = reached – affected**.

How do I reference the various types of lines that SPFLite maintains ?

Because each of these various sets of lines could contain zero or more lines of data, and they might not be contiguous (adjacent) lines, it is very similar to the way SPFLite handles user-defined tags. Thus, SPFLite uses **tag notation** to describe these lists of lines.

The tags for these lines begin with a **Z**, and so they are called "**Z tags**".

- You refer to the list of **reached** lines using the tag :**Z**
- You refer to the list of **affected** lines using the tag :**ZF**
- You refer to the list of **non-affected** lines using the tag :**ZNF**

The list of **eligible** lines for a given command is internally managed by SPFLite. **There is no Z tag for the eligible list.**

What do I need to know about Z tags ?

- A Z tag is a "special" tag defined by SPFLite. You cannot place a Z tag in the sequence area (line-number field) of a data line, either manually, with a key mapping, or with a macro.
- A Z tag is specified on a primary command, just like a "regular" tag that you defined yourself.
- A primary command cannot contain both a Z tag and a regular user-defined tag.
- A Z tag represents a "condition" or "state" of a data line, based on the outcome of a prior command, and is not an actual tag in the conventional sense. That means a data line which is referenced by a Z tag in a chained command could already have a regular user-defined tag. The two uses do not conflict.

When are the affected lines the same as the reached lines ?

When a command like **FIND ABC ALL** is executed, some lines may have **ABC** and some lines may not. So, for any given line, the outcome of the search for the string **ABC** is **conditional**, because it may or may not be found. In such cases :**Z** and :**ZF** may not contain the same list of lines.

For commands like **UC**, every eligible line is converted to upper case, and so it is **unconditional**, since it is not dependent on whether a string is found or not.

If a command is unconditional, the lines reached and the lines affected will be the same. That means in such cases that **:Z** and **:ZF** will contain the same list of lines, and so **:Z** and **:ZF** could be used interchangeably for such commands.

Which command and which lines do Z tags refer to ?

When a Z tag appears on a primary command, it **refers back** to the prior primary command that **immediately preceded** the command where the Z tag itself appears.

For example, consider the following command chain:

```
FIND ABC ALL ; UC :ZF
```

The Z tag **:ZF** refers back to the command which immediately preceded it, which is the **FIND** command.

In this example, SPFLite **creates** the various Z lists for the **current** command, but you **refer** to lists that were created on the **previous** command using the Z tags.

That only makes sense. A Z tag like **:ZF** couldn't refer to lines on the *current* command, because **:ZF** is a list of lines that had been found *already*, and that list of lines is the very place where the *current* command is going to 'look' to perform its operations. If **:ZF** meant lines on the *current* command, it would be referring to "found" lines that hadn't even been found yet.

AZ tag never refers to lines that are reached, affected or unaffected by the command that a Z tag itself appears on.

For example, in the command,

```
CHANGE ABC DEF ALL :ZF
```

the Z tag **:ZF** does **not** refer to any lines changed by this **CHANGE** command **itself**, but by whatever command immediately **preceded** this **CHANGE** command.

A Z tag always "points to the left" to the prior command.

Do the Z tags **:ZF** and **:ZNF** mean Found and Not Found ?

Sometimes, but not always.

- For the **FIND** and **CHANGE** commands, **:ZF** is the list of all reached lines where the search string was **Found**. The **:ZNF** tag is the list of all reached lines where the search string was **Not Found**. (Of course, for **CHANGE**, the command has to **find** a string first in order to **change** it.) For these cases, this may help you to remember which Z tag means what.
- For other types of commands, if you think of the reached lines as the "Z lines", then **:ZF** is the set of "Z lines" that were "Found to be affected" by the command, and **:ZNF** is the set of "Z lines" that were "Not Found to be affected" by the command.

Presently, any chainable commands that take a search string are conditional commands, and will define the **:ZF** and **:ZNF** tags in a way that will mean Found or Not Found. Commands like **UC** that are unconditional also define the **:ZF** and **:ZNF** tags, but they are not important, and you only need to be concerned with the tag **:Z** for these commands.

How is the notion of affected and unaffected lines handled for “Negative logic” commands like NFIND ?

For **NFIND**, the command locates all lines that do **not** contain a search string. So, the object of the **NFIND** command is to **find lines** rather than to **find strings**. That is, the lines located by **NFIND ABC ALL** are the lines that did **not** contain **ABC**.

If SPFLite strictly followed the rules discussed above, lines **not** containing the search string would be the “affected” lines, which would be represented by **:ZF**, rather than by **:ZNF**. For example, the commands **X ALL ; NFIND ABC ALL** would result in unexcluding all lines not containing **ABC**. So, the finding of **ABC**, and the unexcluding affect, **would never occur on the same lines**.

That would have meant that the Z tag **:ZF** would be the list of lines where **ABC** was **not** found, and **:ZNF** would be where **ABC was** found.

Most users would see that as “backwards” and find it hard to understand or remember.

To avoid this confusion, in cases where “Negative logic” commands are used that specify a search string:

- the Z tag **:ZF** will always represent lines where the search string was **Found**
- the Z tag **:ZNF** will always represent lines where the search string was **Not Found**

So, you can always think of **:ZF** and **:ZNF** as representing lines where a search string was either **Found**, or **Not Found**, regardless of whether the command was a “Positive logic” or a “Negative logic” type.

Bear in mind that “Negative logic” commands always require a search string, since they act on lines where the string is not found.

The commands affected by this are **NDELETE**, **NFIND**, **NEXCLUDE**, **NFLIP**, **NREVERT**, **NULINE** and **NSHOW**.

How are **DELETE** and **NDELETE** commands with search strings handled ?

When a command of the form **DELETE ABC ALL** is issued, every line where **ABC** is found is deleted. Since every “affected” line is now gone, **the affected list is empty**. That means the Z tag **:ZF** will have no lines. Any reached lines that were **not** deleted because **ABC was not** found will be in **:ZNF**.

Likewise, when a command of the form **NDELETE ABC ALL** is issued, every line where **ABC** is **not** found is deleted. Since every “affected” line is now gone, the affected list is empty. However, because of the considerations noted above about “Negative logic” commands, the roles of **:ZF** and **:ZNF** are reversed in this command, so that they reflect whether the search string was found or not.

That means, for **NDELETE**, the Z tag **:ZNF** will have no lines, because the command requests that any lines where **ABC is not** found is to be deleted. (There are no longer any “not found” lines, because the command got rid of them all.) Any reached lines that were **not** deleted because **ABC was found** (**that is, lines where ABC can still be found**) will be in **:ZF**.

What happens if a command chain is used out of context ?

The first time you issue a command in an Edit session, since no prior command has been issued, the Z tags **:Z**, **:ZF** and **:ZNF** are empty and do not refer to any lines in your file.

If you attempt to reference a Z tag of **:Z**, **:ZF** and **:ZNF** when it is empty or undefined, you will just get a “not found” or “end of data reached” condition.

Are line commands permitted within a primary command chain ?

Yes. You can intersperse one or more line commands in between primary commands. A Z tag on a primary

command refers back to the last **primary** command, not to any **line** commands you may have issued. The **line** commands do not affect or disturb the way the "chain" operates.

As noted above, commands like **UP** and **DOWN** are not "chainable" but may appear in between primary commands that are chainable. Thus, you can use **UP** and **DOWN** (probably mapped to Page Up and Page Down keys) to locate a line on which you want to issue one or more line commands, before resuming the use of primary commands within a chain.

Can the **LINE** primary command be used in chaining ?

Yes. **LINE** is a regular primary command that accepts a line-control-range operand, and so it can be chained.

Can an unconditional command be simulated as a conditional one for chaining purposes ?

Sometimes. For example, the **UC** command unconditionally converts all lines to upper case. This is "unconditional" in the sense that lines that are already in upper case, lines with no alphabetic data, and even blank lines can be "successfully" converted to upper case.

Suppose you wanted **UC** to be conditional, and you wanted to shift all data lines 4 columns to the right if any data lines contained lower case letters that were converted to upper case letters. Here is how you could do it:

```
FIND P'<' ALL .1 .100 ; UC :ZF ALL ; LINE ')4' ALL :Z
```

The **FIND** locates all lines having any lower case letters. The **UC** converts only those lines with lower case to upper case. The **LINE** command applies the right-shift 4 line command **)4** to all the lines processed by **UC**, which were the lines found to have lower case letters by **FIND**.

Such techniques will work in some, but not all, cases.

Where this cannot be applied, you may need to use traditional tags, **X|NX** lines, **U|NU** lines, or macros to accomplish your task.

If I use multiple commands per command line, where are the messages for them displayed ?

When multiple commands per command line are used (separated by semicolon), the message issued by the **last** command will appear on the edit screen, preceded by a **+** plus sign. If you issue the **HELP** command (usually mapped to the **F1** key) a popup will display the messages produced by each command, in the order issued.

Messages for multiple commands are handled this way regardless of whether the commands involve chaining or not.

Example of command chaining

Suppose you wanted to perform a **CHANGE** on a set of lines, then you wanted to convert all of those lines just changed to upper case, and finally you wanted to shift all of those lines 4 columns to the right. How might you go about such a task?

```
CHANGE ABC DEF .1 .100 ALL ; UC :ZF ALL ; LINE ')4' ALL :Z
```

Note: The **UC** applies to all lines changed by the **CHANGE** command, so **:ZF** is required. Since **UC** is an unconditional command, either **:Z** or **:ZF** would work for the **LINE** command; they would have the same set of lines defined. We use **:Z** here instead of **:ZF** because it is shorter.

Working with the File Manager

Contents of Article

- [*Introduction*](#)
- [*Example of File Manager display*](#)
- [*File Manager screen layout*](#)
- [*Directory refresh*](#)
- [*Specifying the criteria for directory displays*](#)
- [*Extended File Pattern Support*](#)
- [*File Manager primary commands*](#)
- [*File Manager line commands*](#)
- [*File Manager ALL command and File Lists*](#)
- [*Mouse selection of files*](#)
- [*Changing the file display order*](#)
- [*Managing File Lists*](#)
- [*File List Note Support*](#)
- [*Performing a file search*](#)
- [*Find in Files \(FF\) syntax*](#)

Introduction

The SPFLite File Manager is designed to provide a powerful, easy-to-use interface for file selection and file management. This includes a wide range of capabilities to manage file operations, and support for File Lists (collections of files) to provide quick access to frequently used files and directories. server.

Note: The file extension of a File List file is **.FLIST**. When speaking informally about one or more of such lists, we use the more conversational phrase "File List" or "File Lists".

File Manager features:

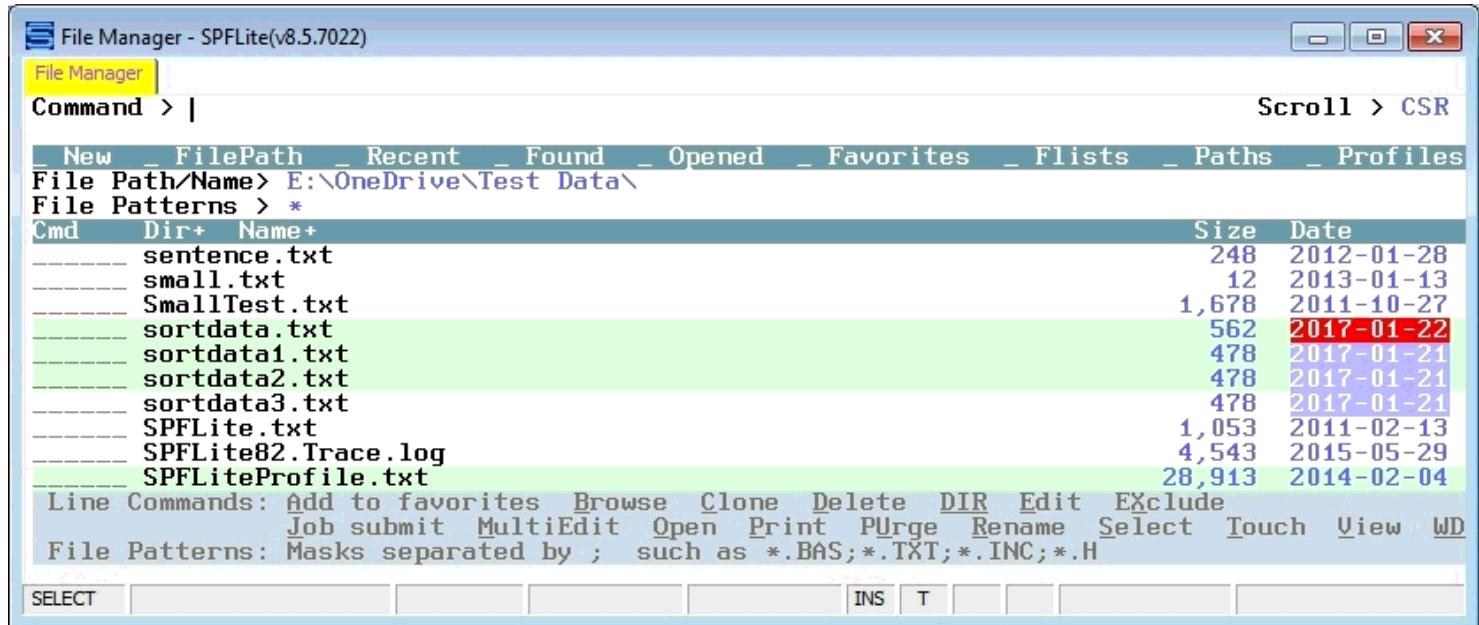
- A variety of predefined File Lists: **Recent Files**, **Favorite Files**, **Open Files**, **FF Result Lists**, **Recent Paths** and **Profile Lists**.
- Directory path lists
- Extensive File Pattern matching can be used on the file names to display only wanted files.
- Large scrollable line-command field can be used to enter long line-command names, or commands with arguments (like **Add** and **Rename**). This field is padded with `_` characters. Note that if the File/Path being displayed is flagged as Read-Only, the field will be padded with `.` i.e. `.....` instead of `_____`
- A Global Options tab is available for the File Manager. This gathers all global options relating to the File Manager into a single dialog, which makes them easier to find and modify as needed. Enter **OPTIONS FM** to reach these settings.
- The **ALL** line command can be used to apply selected File Manager line commands to every file named in a File List. See discussion below for more information.

File Manager options are set in the [Options - File Manager](#) settings page:

- When you check [Close SPFLite with last file tab](#), SPFLite will terminate when the last active Edit/Browse Tab closes. If unchecked, and no file tabs are open, SPFLite continues running and the File Manager remains open.
- When you check [Confirm file Deletes](#) a popup message will appear when you request a file to be deleted, giving you a chance to confirm that you are deleting the right file. **For data security reasons, we recommend you enable this option.**

- When you check [Display File Manager Help](#), a Help legend appears at the bottom of the File Manager screen, showing available File Manager line commands that can be used. Depending on whether you are viewing a directory list, a **Recent Files** File List or a **Named Favorites** File List, different help legends with different line-command codes are shown. Because different help legends are displayed, and different features are supported on the various screens of the File Manager, most users will benefit from enabling the File Manager Help display. (This is especially so in view of the fact that the File Manager gets progressively updated with new features, which you might not otherwise be aware of.) If you need more room on the File Manager screen and can remember the various codes, this option can be unchecked to regain some space.

Example of File Manager display



File Manager screen layout

The top part of the File Manager display contains 4 lines which will always be present, even during scrolling operations of the displayed list. These lines are:

Quick Launch Bar

This line contains mouse click-able options to provide fast switching between common selections. For keyboard fanatics, each option also has a single character input field to its left; you can select an option by entering any non-blank character in this field. The Quick Launch options are:

- New** Requests opening a new, empty Edit tab. Use this for creating a new file from scratch.
- FilePath** Requests a display of files based on the **File Path/Name** and **File Patterns** values entered on the 2nd and 3rd lines of the header area.
- Recent** Request a display of recently accessed files
- Found** Requests a display of the latest results from an FF (Find in Files) search command.
- Opened** Requests a display of all files currently opened for processing in other tabs.

Favorite	Requests a display of the list of files you have identified as your 'favorite' files. These are identified using the FAVORITE command or the File Manager ADD line command.
Lists	Requests a display of all your saved File Lists.
Paths	Requests a display of recently used File Paths.
Profiles	Requests a display of all your current Edit Profiles.

File Path/Name
File Patterns These lines provide the criteria for a Directory folder display. See [Criteria](#) for details.

Column Header This provided headings for the file list display as well as being a click-able selection line to change the sort order of the display. See [Changing the file display order](#) for details.

Following these header lines will be the detailed list display you have requested. The left side of each line always contains a Line Command input field followed immediately by the filename. The remaining columns of the display are under your control, you can select what other columns to display, and what order they will be in from left to right. See [Options - File Manager](#) for details.

This portion of the display is scrollable when needed using PgUp/PgDown, Mouse scroll wheel, or keyboard Up/Down keys.

If **HiLite Recent / Active Dates** has been activated, the dates in the Date column will be hi-lighted appropriately. Note the dates in the example hi-lighted in Red and Blue. See [HiLite Recent / Active Dates](#) for more information.

Directory refresh

Once File Manager has displayed a directory, it will refresh the display with the results of any file activity the next time the File Manager screen is displayed, either by a swap between File Manager and some other tab, by performing some action on the File Manager screen itself, or just pressing Enter.

Specifying the criteria for directory displays

When you are displaying a directory, the header portion of the File Manager screen contains input fields where you specify the directory you wish to be displayed and what file types should be included.

In the **File Path/Name** field, enter the directory name you wish to be displayed. This may be a simple drive letter like **c : ** or a subdirectory like **c : \MYDIR**. Path names with embedded blanks are simply typed as-is, with no quoting required. A trailing backslash in a path name should be specified, without the \ the name will not be treated as a path request.

The **File Patterns** field should contain a list of one or more file patterns or "wildcards" of files you wish to display. The format used is the based on that used by Windows Explorer or the command line.

A ? question mark matches any single character.

An * asterisk matches zero or more characters up to the **next** specified mask character (or end of filename).

A double asterisk ** matches zero or more characters up to the **last** occurrence of the next specified mask character. For example **** .** would mimic Windows Explorer handling of *** .** which skips to the **last** period in the

name.

To display all files in a directory, specify ***** or ***.*** as usual. Multiple file patterns are allowed, each separated by a semicolon. Example: ***.BAS ; * .TXT ; * .INC ; * .H**

The initial sort order is by ascending file Name. See [Changing the file display order](#) below for more information.

The last-used File Path, File Patterns and Sort criteria in effect when SPFLite is terminated are remembered and used for the opening display the next time SPFLite is started. So, if you change the sort order to descending by size, for instance, it will be that way the next time.

Extended File Pattern Support

SPFLite supports an enhancement to File Patterns to provide extended flexibility in file selection when needed.

Normal file patterns are inclusive; that is, they specify what files to **include**. You can also specify what files to **exclude** from the selection process. This is done by preceding an exclusive mask string with a **-** minus sign. For example, you can say **-*.INC** to exclude **.INC** files from the list.

Inclusive masks can optionally include a leading **+** plus sign, but this is implied and not needed.

How does file inclusion and exclusion work? A file is processed against the File Mask string from left to right, and each pattern in the File Mask is tested against the file name for a match. An overall match result (true or false) is created based on the individual tests.

Note: Even if one test in a chain of tests fails, the remaining tests are still evaluated. This makes it possible to selected files based on rules that amount to, "I want files of one type, except when they are of a second type I don't want, unless those files are of a third type that I really do want."

Inclusive Masks: If matched, the overall match result is true. If not matched, the match result flag is left unchanged.

Excluding Masks: If matched, the overall match result is false. If not matched, the overall match result is left unchanged.

Examples:

File: TestFile.TXT
me:

Mask: ***.TXT ; -Test*.***

Result: The file would **not** be selected. It is accepted by ***.TXT** but is rejected by **-Test*.***. The final overall match result is false.

File: TestFile.TXT
me:

Mask: ***.TXT ; -Test*.* ; *File.***

Result: The file **would** be selected. It is accepted by ***.TXT** but is rejected by **-Test*.***; but then is accepted by the ***File.*** mask. The final overall match result is true.

Note: If you begin a File Pattern with an exclusion mask, like **-*.tmp**, SPFLite will internally prefix this with an *****; since beginning with an exclusion mask is illogical.

Set Symbol support in File Patterns

The File Patterns string may now include SET symbols for cases where you may wish to save complex mask

strings as symbols for easier use when needed. They are used as replacements for masks as follows: (assume set symbol ABC=*. * and set symbol DEF=*. TXT)

For inclusive masks enter as =abc which would be treated as *. *, or if you prefer explicit inclusive indication +=abc

A mask of =abc; -=def would be treated as *. *; -*. txt

File Manager primary commands

The File Manager supports the following primary commands, any of which can be mapped to a key using the KEYMAP facility. Some File Manager primary commands have the same name as Editor primary commands, but they do not all perform the same function. Refer to the list below for a complete description of each command's function.

ALL	Allows a request to perform a specified Line command against all displayed filenames in the current list. e.g. ALL MEDIT would open all listed files in a single MEdit session
BOTTOM	Scroll to the bottom of the directory or File List display. BOTTOM can also be spelled as BOT .
BROWSE	Browse a file, or without a file name, open a Browse dialog window. To browse a named file that has embedded blanks, enclose the name in quotes. BROWSE can also be spelled as BRO or B .
CASE	To set the default CASE to be used by the FF command.
CLIP	Open an edit session to directly edit the contents of the Windows Clipboard.
CLONE	Clone a copy of an existing file. You must specify the filename operand when using this command in File Manager
CMD	Used to execute an external program or script.
CRETRIEV	Conditionally move the cursor to the Home position or Retrieve the last saved command from the Retrieve Stack.
DEFAULT	Normally, a left-click on a File Manager line item, will be interpreted as an S or SELECT line command. You may change this default command assumption using the DEFAULT command. e.g. DEFAULT BROWSE would cause a left-click to Browse the selected file. The setting is not permanent and is not remembered between SPFLite sessions.
DOWN	Scroll the display down by the amount shown in the Scroll amount field. Typically mapped to the PageDn key.
EDIT	Edit a file, or without a file name, open an Edit dialog window. To edit a named file that has embedded blanks, enclose the name in quotes.
END	Close a directory or File List display, returning the display to its parent directory or to the File List that had previously been displayed. Once a directory list is at a root directory like C:\ the END command will have no further effect.
EXCLUDE	Exclude files from a File List display based on a mask style operand.

FF	The Find in Files command FF searches all currently displayed file names for a string value. See Performing a file search below.
	Do not confuse the File Manager's Find in Files command FF with the Edit/Browse command FF , which is an alias of the familiar FIND editor command. The two commands are merely spelled the same but are unrelated.
FIND	The FIND command in File Manager searches the displayed file list for a simple string argument, handy to locate a file in a long directory list. Only simple search strings are supported, not the special string types such as P'xx', R'xx' or X'xx'.
	Do not confuse the File Manager's FIND command with the Edit/Browse command FIND . The two commands are merely spelled the same but are unrelated.
HELP	Display the SPFLite Help facility. HELP followed by a command name displays the SPFLite Help facility and positions the display at the Help information for that command.
KEYMAP	Brings up the SPFLite KEYMAP facility
LOCATE	LOCATE in File Manager is used to quickly locate the file called <i>name</i> in the file list, and may only be used when the list is in Name+ or Name- order. If no files are found that exactly match the <i>name</i> operand, LOCATE will try to find the first file name in the list that begins with <i>name</i> . If there are no files that match <i>name</i> or begin with <i>name</i> , LOCATE will find the first file that is greater than <i>name</i> . LOCATE can also be spelled as LOC or L .
	Note that the File Manager LOCATE command is unrelated to the editor LOCATE command. See also the FIND command above.
MAKELIST	Create a new File List with a new name, based on the files that currently appear on a directory display or another File List display. MAKELIST can also either completely replace an existing File List with a new set of filenames, or you can request a merge of the current list of files with those already in an existing File List. See the REPLACE and APPEND operand descriptions in MAKELIST . MAKELIST can also be spelled as ML .
OPEN OPENV OPENB	Open a file in a new SPFLite instance. To open a named file that has embedded blanks, enclose the name in quotes. The OPENV and OPENB variations can be used to open the new session in View or Browse mode, instead of the normal Edit mode if desired.
OPTIONS	Bring up the SPFLite Global Options window. See OPTIONS - Set Global Editor Options for more information.
PRINT SETUP	The PRINT SETUP command may be launched from File Manager or from an edit session. PRINT without the SETUP keyword may only be issued from an edit session, not from File Manager. To print a file from File Manager, use the P line command.
RESET	Resets the hidden status of Excluded and/or Forgotten files in a File List
RETF	Retrieves the 'next' saved command from the Retrieve Stack, in a forward direction, opposite to that done by the RETRIEVE command. RETF can be used when you are looking for a particular saved command using RETRIEVE and you inadvertently 'pass up' the one you wanted; you can then use RETF to 'go the other way' to get the passed-up command. If you continue to issue RETF commands (usually from a mapped key like Shift-F12), successively newer entries are retrieved in a forward direction.

RETRIEVE	Retrieve the last saved command from the Retrieve Stack. If you continue to issue RETRIEVE commands (usually from a mapped key like F12), successively older entries are retrieved in a backward direction.
SET	SET brings up the SET Variable Editor.
SWAP	SWAP PREV and SWAP NEXT are used to select the previous or next tab on the SPFLite display, which may be the File Manager tab or an edit session tab. Other SWAP options are also available. See SWAP - Switch to a Selected File Tab for more information.
TOP	Scroll to the top of the directory or File List display.
UP	Scroll the display up by the amount shown in the Scroll amount field. Typically mapped to the Page Up key.
VIEW	View a file, or without a file name, open a View dialog window. To view a named file that has embedded blanks, enclose the name in quotes. VIEW can also be abbreviated as V .

File Manager line commands

Next to each line entry in the File Manager display is an underlined area where you may enter commands to be performed against the line entry. The command entry area is a scrollable field, so there is no restriction on the full length of any command entered. In addition, the width of the File Manager line command field can be adjusted, by setting a column width for it on the [File Manager Global Options](#) dialog. However, most FM line commands may be entered as simple single character values.

Lines with names ending in \ backslash are directories, where . . \ means the “parent” directory, using the common notation for this. Directories can be selected by a mouse left-click or with the line command **S**, which will cause the File Manager list to be refreshed with the contents of the selected folder. If you select multiple directories at once, only the last (lowest on the screen) will be used. Only a limited number of commands can be used on directories.

Many File Manager line command fields are much longer than the 1-character codes used in prior versions. This allows commands like **ADD** and **RENAME** to take an operand field, and it also allows for future enhancements to the File Manager.

If you enter a line command that takes an operand, the line command field will scroll left and right to accommodate as large of an operand (like a file name) as you need.

Multiple Line Selection

As of Version 8.0, you may select multiple lines in File Manager much as you select multiple lines in a normal Edit screen. All the commands which have a single character abbreviation (**A**, **B**, **C**, **D**, **E**, **F**, **J**, **M**, **N**, **P**, **R**, **S**, **T**, **U**, **V**, **X**) can select multiple lines either by using the paired block mode technique (**MM / MM**, **SS / SS**, etc) on the first and last lines of a range; or by adding a 'repeat' number to the end of the command (**S4**, **D3**, etc).

The following line commands are available:

.Profile-name	If a Profile override operand alone is provided as a line command, it will be taken as a request for an Edit of the file using the specified Profile override. i.e. it is a shortcut for entering EDIT .Profile-name
A [<i>file-list-name</i>]	Add to favorites. If an optional file-list-name is provided, the name of the file is added to the specified Named Favorites File List; otherwise it is added to the
AA	

ADD

default (unnamed) **Favorite Files** File List. You cannot add directories or other File List names to a File List.

No harm is done if you attempt to add a file to a Favorite File List if it's already there. It just "confirms" that the file name has been added.

A given file can be recorded in as many different File Lists as you wish. To avoid creating a confusing situation, it is usually best to limit the number of File Lists a given file is recorded in.

There is also available an **AUTOFAV** facility where files can be automatically added to Favorite lists. See "[Using AUTOFAV to create FILISTS](#)".

ALL *command*

The **ALL** line command can only be applied to a File List. *command* is a selected File Manager line command. SPFLite reads the File List to determine a list of files, and it applies *command* to each file. See discussion below for more information.

B [.Profile-name]
BB
BRO
BROWSE

Browse the file. In Browse mode, **NO** changes can be made to a file whatsoever. Browsing a File List is permitted. See the **E** command below for more information. Note that you can use **ALL B** to **Browse** all the files listed in a File List.

If the Profile override operand is provided, the Browse will be started using the provided Profile name rather than the one implied by the file's extension.

CANCEL | CAN

The **CANCEL** command is only allowed to be used on files displayed by the Open Files display. It executes a **CANCEL** command for the specified file and returns to the File Manager Display

C [.Profile-name]
CC
CLONE

Clone the file. Cloning a file provides a convenient way to create a copy of a file for editing. You will be prompted for a new filename for the cloned file. An optional Profile name may be entered if you wish to open the cloned file with a non-standard Profile name.

A directory cannot be cloned with the **C** line command, but you can accomplish essentially the same thing by using the **MAKELIST** command on a directory. See [MAKELIST - Create FILELIST](#) for more information.

D
DD
DEL
DELETE

Delete a file or directory. File deletion is protected by two features in SPFLite. The first is the File Manager Options checkbox [Confirm File Deletes](#). If checked, a popup message will ask for confirmation before the file deletion is dealt with.

The second is the General Options checkbox [Delete to Recycle Bin](#). This option controls the kind of deletion that occurs. If checked, a popup message will ask for confirmation before the file is Recycled. If unchecked, a popup message will ask for confirmation before the file is Purged. The term **purge** means permanent deletion with no recourse.

Where SPFLite can detect it, the Delete command will also remove deleted file names from File Lists in a manner comparable to the **Forget** line command. If a **Delete** or **Forget** command causes all names to be removed from a File List, the File List itself will be deleted. If you delete a file which is named elsewhere in another File List, it will be removed from that File List the next time that File List is opened (unless the file name is "guarded").

Deleting a File List is permitted.

A directory can be deleted with the **D** line command, but only if it is empty.

If the **DELETE** command is issued against a file displayed under the Open Files display, it will execute a **CANCEL DELETE** command for the specified file and return to the File Manager Display

DIR

The DIR command will switch the File Manager display to the folder containing the filename of this line using a File Pattern of *.*

E [.Profile-name]

EE

EDIT

.Profile-name

Edit the file. Clicking on a file name is the same as entering the **E** or **S** line command, and causes the file to be edited.

When **E** is used on a File List, SPFLite brings up an Edit session on the File List file. A File List file is an ordinary text file managed by SPFLite, with one fully-qualified file name per line. You may add, change or delete lines in this file. Since these names are supposed to represent valid file names, you should edit this file carefully.

File names in a File List which are Guarded will be shown preceded by a | character.

One reason to edit a File List is to add wildcard file name entries with ? and * codes; another reason may be to import a large number of file names into a File List from an outside text file. Because the **Recent Files** File List is automatically maintained by SPFLite, editing it is not recommended. A directory cannot be edited.

Note that you can use **ALL E** to **Edit** all the files listed in a File List.

If the Profile override operand is provided, the Edit will be started using the provided Profile name rather than the one implied by the file's extension.

If you click on a file name to invoke the Edit function, you can enter the .Profile-name by itself on the FM **line command** area, then click on the file name, without having to use the **E** or **EDIT** command.

END

The **END** command is only allowed to be used on files displayed by the Open Files display. It executes an **END** command for the specified file and returns to the File Manager Display

F FF FORGET

Forget the file. Applies only to entries created by a File List. The Forget operation 'hides' the line in the File List containing the file in question. When files have been 'forgotten' from a list, the total line at the bottom of the list will display the count of 'forgotten' files. Forgetting a file only hides an entry from a File List; it does **not** delete the file.

The Forget action is remembered and the next time the File List is displayed, the forgotten files will still be hidden from the display.

Unlike SPFLite versions prior to Version 8.0 you may now Forget files even if they appear in a File List display as a result of a wildcard specification within the File List file.

J

Job Submit. The file is "submitted" using existing user-defined **SUBMIT**

JJ
JOB
SUB
SUBMIT

configuration settings as specified on the [Options – Submit](#) screen. The **SUBARG** value, if any, defined for the file type of this file, will be properly defined as set up in the file type's PROFILE information. The action of the **J** command is the same as if the file were opened in an Edit session, and then the **SUBMIT** command were issued with no arguments.

There is no direct way to debug a job submitted in File Manager with the **J** command; there is no equivalent of the **DEBUG** operand used by the **SUBMIT** command. If you need to debug a job submission using the **J** command, the easiest way to do this is to specify a batch file name as the Submit Prototype command. In your batch file, you could issue a **PAUSE** command to see any output produced, or take other debugging actions.

L
LL
LINE
LINES

Lines - create or update the STATE information which is used to provide the information shown in the LINES column of the File Manager display. This command will update that information without requiring a full opening of the file.

Note: The Profile for the selected file's file-type must have the profile option of **STATE ON** set. If not an error message will result.

M
MM
MEDIT

Multi-Edit. This is the ability to perform a multi-edit session. A [multi-edit session](#) exists when you edit multiple files, at the same time, in the same edit tab. You begin a multi-edit session by selecting one or (usually) more files from a directory or File List display using the **M** line command. You can scroll up and down in the list to find all the files you want to place **M** codes on, but don't press Enter until you have chosen all of them. Once you have decided upon the files to be multi-edited, press Enter. An edit session will appear with "**(M-Edit)**" as the label for the tab. Each file will be preceded by a 'marker line' with **=FILE>** in the sequence area and the file's name next to it. When you **SAVE** or **END** a multi-edit session, every file in the session will be saved at the same time. See [Working with Multi-Edit Sessions](#) for more information.

Note: you can use **ALL M** to start a **Multi-Edit** session using all the files listed in a File List.

N
NN
NORM

Forget support for File Lists no longer actually deletes the forgotten files entries in the File List dataset. Similarly, displaying a File List which has entries in it for specific files, and those files no longer exist, does not trigger their removal from the file list, they are simply ignored. The **NORM** (normalize) line command, used against a File List, will perform this periodic cleanup function.

OPEN / O
OPENV / OV
OPENB / OB

Open a file in a new SPFLite instance. The **OPENV / OV** and **OPENB / OB** variations can be used to open the new session in View or Browse mode, instead of the normal Edit mode if desired.

P
PP
PRINT

Print the file. The file is immediately sent to the printer using the currently configured printer settings, as defined by the **PRINT SETUP** command. Note that the listing is produced unnumbered (the default), because since you cannot enter the **NUM** operand that is available with a **PRINT** primary command.

Files printed via this method are always printed in non-color mode. To print in full colorize mode, you must issue a **PRINT** command from a normal Edit/Browse tab.

R	Rename a file or directory. You may specify the new filename directly as an operand to Rename ; if omitted, you will be prompted for the new name.
[<i>newfilename</i>]	
RR	
REN	When you rename a file, if it is recorded within any current File Lists, the entries within those File Lists will also be renamed.
RENAME	
	Renaming a File List is permitted. If you wish to save a Recent Files File List or Found Files File List once a set of files has been stored, you can rename it to some other name that ends in .FLIST . Then, the list will appear in the list of Named Favorites .
	Note: Do not confuse the File Manager line command REN with the editor primary command REN . In File Manager, REN means RENAME .
	In SPFLite prior to release 8.5, the editor primary command REN was an abbreviation for the command RENAME . As of SPFLite version 8.5, the editor primary command REN is now an abbreviation for the sequence numbering command RENUM . This change was made for ISPF compatibility.
SAVE	The SAVE command is only allowed to be used on files displayed by the Open Files display. It executes a SAVE command for the specified file and returns to the File Manager display
S [<i>.Profile-name</i>]	Select the file. For data files, Select is the same as Edit. Clicking on a file name is the same as entering the E or S line command, and causes the file to be edited.
SS	
SEL	
SELECT	For directories and File Lists, Select will open the directory or File List and its contents will replace the currently displayed list. Thus, for a File List, Select and Edit are not the same thing , whereas for regular files, they are .
	If the Profile override operand is provided, the Edit will be started using the provided Profile name rather than the one implied by the file's extension.
	If you click on a file name to invoke the Select function, you can enter the <i>.Profile-name</i> by itself on the File manager line command area, then click on the file name, without having to use the S or SEL command.
T	
TT	
TOUCH	Touch the file. The file's timestamp is replaced by the current date and time. A Touch operation is commonly needed by users of MAKE and BUILD software. This operation is not allowed on directories and File Lists.
U	
UU	
PUR	
PURGE	Purge (delete) the file. File deletion is protected by two features in SPFLite. The first is the File Manager Options checkbox Confirm File Deletes . If checked, a popup message will ask for confirmation before the file deletion is dealt with.
	The second is the General Options checkbox Delete to Recycle Bin . This option controls the kind of deletion that occurs. For the Purge line command, the Delete to Recycle Bin option is ignored. In all cases, a popup message will ask for confirmation before the file is Purged. The term purge means permanent deletion with no recourse.
	Where SPFLite can detect it, it will remove purged files from File Lists in a manner comparable to the Forget line command. Purging a File List is permitted.
	If the PURGE command is issued against a file displayed under the Open Files

display, it will execute a **CANCEL PURGE** command for the specified file and return to the File Manager display

V	[.Profile-name] View the file. In View mode, changes can be made to a file but you cannot save them. Viewing a File List is permitted. See the E command above for more information. Note that you can use ALL V to View all the files listed in a File List.
]	
VV	
VIEW	If the Profile override operand is provided, the Browse will be started using the provided Profile name rather than the one implied by the file's extension.
WDIR	Open a Windows Explorer window for the folder containing this file. The selected file will be highlighted in the Explorer display.
X	
XX	Exclude (X) is available for hiding files from the File Manager display. Visually this looks the same as Forget , but Excluded lines are not 'remembered' between sessions, the effect lasts only as long as the current list display is shown.

File Manager ALL command and File Lists

You can apply selected File Manager line commands to a File List, and the commands will be applied, not to the File List **itself**, but to the files named **within** the File List. For example, to edit all of the files named in a File List, you would issue the File Manager line command **ALL E** for that File List.

The **ALL** command is followed by one of the following commands (all valid abbreviations of these commands are allowed):

BROWSE
CLONE
DELETE
EDIT
MEDIT
NORM
PRINT
PURGE
SELECT (same as **EDIT**)
TOUCH
VIEW

WARNING!

The ALL command, if used against a File List which contains generic path requests, could effectively be directed at what might be hundreds of files. Even if you do not normally receive prompts for file deletes, you will still receive **one single prompt** for the **ALL D** command. But a single wrong reply and you could have a **huge** problem. Take care!

Mouse selection of files

You may select a file for editing, or a directory or File List to be displayed, by clicking on its name with the left mouse button. If you click on any of the underscore characters next to the name, the cursor will be moved to the left hand column of the line command area. This is a quick way to get the cursor moved so you can enter a File Manager line command.

If you click on a name it will be treated as if the Select line command **S** were used. This default of **S** can be altered using the **DEFAULT** command.

For files, the mouse-select action is the same as the Edit line command **E / EDIT**, and the file will be opened in a

new edit tab.

For directories and File Lists, the mouse-select action opens up the directory or File List, and its contents replaces the current display. To return from a lower-level directory to its parent, or to return to the prior display from a File List display, use the primary **END** command (traditionally mapped to F3), or right-click on the File Manager tab.

Changing the file display order

When specifying the sort criteria below, the settings are saved as defaults for the **current** FM display type. e.g. Recent, Favorites, Paths, etc. Separate sort criteria are saved for each of the display types chosen via the Quick Launch bar. This allows you to maintain unique preferences for the differing list types.

The default sort criteria is to sort ascending on filename. The sort order may be changed at any time, by clicking on the column heading for the column you wish to sort on. These column headings are:

Dir	Specifies where Directory lines are to be placed (Top, Bottom, or In-line)
Name	sorts on file name
Note	sorts on the Note data
Ext	sorts on file name extension
Size	sorts on file size in bytes
Date	sorts on the file's date and time

The **Dir** heading does not actually alter the Sort criteria or direction, but is located here for convenience. When clicked, it will 'rotate' through three settings:

Dir+ which requests directory entries are to appear first in the list.

Dir- which requests directory entries are to appear last in the list.

Dir* which requests directory entries be placed alphabetically in the list.

If you click on the other column headings, you will see a **+** code to indicate that column is in ascending order, and a **-** sign to indicate that column is in descending order. For the Name column, this will appear as **Name+** or **Name-**. When no code is present on a heading, the list is being sorted by some other column.

Because the most common reason to display files in **Date** order is to find the most recent ones, clicking on the **Date** column when not already active will first show **Date-** and the list will be in descending date/time order starting with the most recent. Clicking on the heading again will show **Date+**.

Similarly, because the most common reason to display files in **Size** order is to find the largest ones, clicking on the **Size** column when not already active will first show **Size-** and the list will be in descending size order starting with the largest. Clicking on the heading again will show **Size+**.

When a given column is being used as the sort criteria, the data in that column appears in the high-intensity text color, rather than the standard low-intensity text color.

Managing File Lists

The next major topic [Working With File Lists](#) provide an extensive description of File Lists and their creation and management.

File List Note Support

Support is provided for maintaining a Note field for the files displayed by a File List. Notes are intended to provide a small reminder or tracking field associated with each file in a File List. Entirely optional, you can use the Note field for any purpose which may be useful to you. To activate Note support, you must add the Note

column to the optional fields displayed by File Manager. See [Options - File Manager](#) for how to specify this.

Performing a file search

The Find in Files command **FF** searches all currently displayed file names for a string value. You can search a directory list, the **Recent Files** File List or any other File List.

The **FF** command searches every file that is displayed in the current list. To search a 'selective' list of files, you can start with a directory list and use the **File Patterns** field with one or more wildcards to reduce the number of displayed files. You can also add files to a favorites list with the **A** line command or the **FAV** edit primary command, and then issue the **FF** command while the **Favorite Files** File List is displayed. If you search a File List, you can remove files from the list with the **Forget** line command **F** before doing the **FF** command.

If the **FF** command finds at least one matching file, it creates (or updates) the **Found Files** File List and then displays it. When displaying a Found Files list, the **FF** search command used to create the list will be shown next to the Found Files title on the File Manager display. This will remind you how the Found list was created if you should open it at a later time than its initial creation.

You can issue the **FF** command using the **Found Files** File List itself. If you do this with different search strings, you will successively refine the list. That is, if you say FF ABC, then FF DEF, then FF XYZ, you will end up with a **Found Files** File List that only shows files that contain all three strings ABC, DEF and XYZ. Each time you use a different search against the Found Files File List, the list will get recreated and will (usually) be shorter than it was before.

Of course, if you use an **FF** command against the **Found Files** File List with the **same** search string you used to create it, you would just be searching the files you **already found**, and it would **re-find the same files all over again**. That would be called 'going no where fast' (or as a friend used to describe it, "the department of redundancy department").

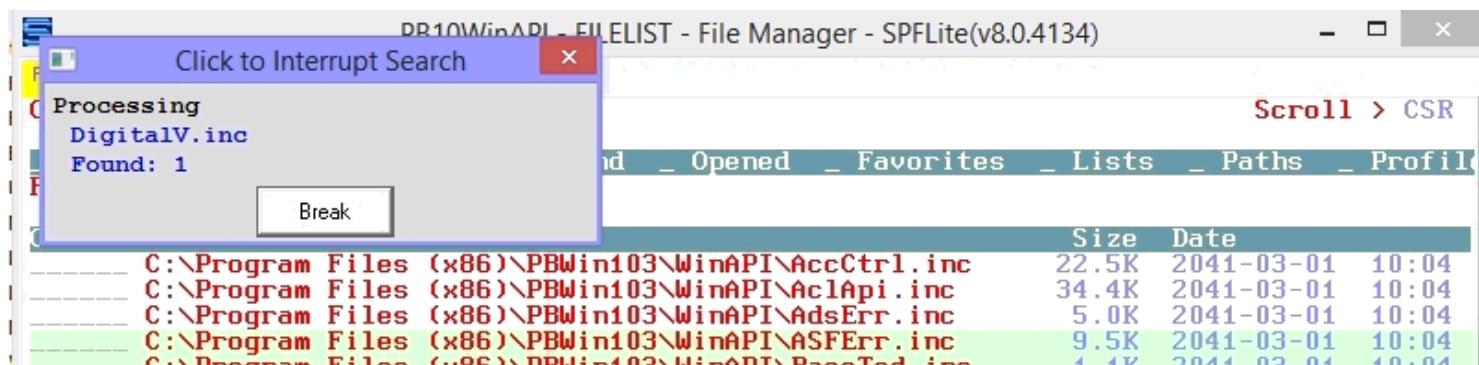
You really want to use **different** search strings each time, unless some outside process is changing the files and you truly need to reconfirm your prior results. That might be a legitimate need, but only in very rare cases.

The **Find in Files** command **FF** uses a syntax similar to (but simpler than) the editor **FIND** command.

Note: the Edit primary command **FIND** has an alias of **FF**. The Edit command **FF** has no relation to the File Manager **FF** command.

Interrupting an FF search

When the FF command commences a search, it displays a small pop-up dialog:



which shows the ongoing progress of the search. You may interrupt the search at any point by left-clicking on the 'Break' button on this dialog. The FF command will complete processing with whatever located files it has already found and display the Found Files list.

Find in Files (FF) syntax

FF *string* [CHARS | WORD | PREFIX | SUFFIX] [NF]

Operands:

string

Any string value accepted in an edit session is permitted here. The *string* may be unquoted, or quoted without a string type, or may be quoted with a string type of C, T, X, P or R. Unquoted strings, or strings quoted without a string type, are assumed to be of type **C** or **T**, depending on the C/T case indicator on the status line. If the status line indicator shows **C** or **C W**, a case-sensitive search is done, and if **T** or **T W** appears, a case-insensitive search is done. You can use the **CASE C** or **CASE T** command in File Manager to change the C/T indicator.

CHARS | WORD | PREFIX | SUFFIX

Specifies the 'search context' for the string, the same as is done for the **FIND** command in the editor. If not specified, either a **CHARS** search or a **WORD** search is done. If the status line indicator shows **C W** or **T W**, a **WORD** search is done, and if just **C** or **T** appears, a **CHARS** search is done. You can change the default search context with the **FIND WORDS** or **FIND CHAR**S command.

NF

If the Not Found option **NF** is used, a search is made for files that do **not** have the *string* present on any line.

Working with File Lists

Contents of Article

[Introduction](#)

[Creating and manually editing File Lists](#)

[General features of File Lists](#)

[Creating File Lists with the MAKELIST command](#)

[The Recent Files File List](#)

[The Recent Paths File List](#)

[The Favorite Files File List and Named Favorites](#)

[The Found Files File List](#)

[The Open Files File List](#)

[File Manager ALL command and File Lists](#)

[File List Cleanup and Forgotten Files](#)

Introduction

File Lists provide a number of new capabilities for managing files. In addition to the standard directory list of files that appears under the File Manager tab, there are new, folder-like entries saved as a file type of **.FLIST** in the SPFLite data directory.

Note: The file extension of these files is **.FLIST**. When speaking informally about one or more of such lists, we use the more conversational phrase "File List" or "File Lists". There is no plural keyword or extension called "FILELISTS" as such.

These File Lists store the names of files of special interest to the user, so they may be found and opened quickly.

- The **Recent Files** File List holds a list of recently opened files, up to a maximum number of files as specified by a global Option value. As more files are opened, files opened further in the past drop off the list once the list reaches its maximum size.
- The **Recent Paths** File List holds a list of recently referenced file path names (just the fully-qualified directory names, not the names of any data files).
- The **Found Files** File List holds a list of files found by the Find in Files command **FF**.
- The **Favorite Files** File List holds a list of files specifically named as "favorite" by the user with the **FAV** command, and do not automatically drop off the list.
- In addition to the "standard" **Favorite Files** File List, users can create their own **Named Favorites** File Lists and add files to such named lists with a **FAV list-name** primary command in an edit or browse session, or by an **Add** line command in File Manager.
- The **Open Files** File List contains the file names of every file currently open in an Edit or Browse tab.

An **FLIST** file is a simple list of requests for files to be displayed together by File Manager. Any number of requests are allowed, and the requests can be for single specific filenames, for all files in a folder, or for subsets of files in a folder based on Filename masking criteria. Requests of all types can be intermixed within a single **FLIST**.

A individual file request can be in more than one list if desired, each **FLIST** is independent of any other.

Creating and manually editing File Lists

FLISTs are simple text files containing one file request per line. **FLIST** files are stored under **My Documents\SPFLite\FileLists** and contain **.FLIST** as their file name extension. Because these are ordinary text files, they can be opened with other editors, and **FLIST** compatible files could be created from outside sources and copied into the **My Documents\SPFLite\FileLists** directory if desired. A File List file can also be edited directly from SPFLite by putting an **E** line command on the File Manager line containing the File List entry.

Editing a File List

Although an **.FLIST** file can have multiple fields per line, other than the first they are all optional. If you wish to make a File List consisting of filenames you have collected from some other source, simply create the file with one fully qualified file name per line and save it as an **.FLIST** file type in the SPFLite FILELIST folder.

If you are Editing an existing File List, and you see these extra fields, you should not normally alter or remove any of them unless you are confident you understand the following detailed description. If you simply want to add new specific filenames, just add them as new lines with fully qualified names.

A File List file can contain any needed number of request lines, and they do not have to be in any specific order. Each line can be for a single specific filename, or it can be a path/folder request. Each path/folder request can also have it's own unique file mask specification.

.FLIST request format

The request line syntax is:

File Path/name[|File mask[|Flags[|Note]]]

Note: The **|** in the syntax above is used as a field delimiter, **not** as an OR indicator. e.g. an entry with three fields would appear as **filename|*|F**

where:

File Path/Name This should be either a fully qualified filename, or, A path/folder name ending in a \

File Mask This should be a File Mask to use with the specified File Path/Name to determine which files in the folder are to be selected. The syntax can be found in [Extended File Pattern Support](#). If omitted, an ***** is assumed.

Flags This field is used internally by SPFLite. If editing an existing **.FLIST** file, do not modify this field; if creating a new entry, omit the field.

Note This field is used to store Note data entered for a File List entry. If editing an existing **.FLIST** file, do not modify this field; if creating a new entry, omit the field. Details on Note support can be found in [File List Note Support](#).

One reason to edit a File List file is to manually create lists of files containing wildcards. Suppose you had three files called C:\MYPATH\ABC1.TXT, C:\MYPATH\ABC2.TXT and C:\MYPATH\ABC3.TXT. Perhaps you expect to have more such files in the future, and you wanted to 'gather' all of them under one named favorites list called ABC.FLIST, even files that do not yet currently exist, without having to manually add them to the favorites list later. How could you do this?

Here is the easiest way:

- First, open C:\MYPATH\ABC1.TXT
- In the edit tab, issue the primary command FAV ABC.

- You will see the message, "File added to ABC.FLIST". This creates the initial File List file in the location where SPFLite will find it.
- Click on the File Manager tab.
- Click on **Lists**.
- Notice that ABC.FLIST appears. Now, place an E line command on the ABC.FLIST line and press Enter.
- The file **My Documents\SPFLite\FileLists\ABC.FLIST** will appear in an edit session, and there should be a line of data containing C :\MYPATH\ABC1 . TXT ||| as text.
- On that line, change the "1" to an "*" and then save the ABC.FLIST file.
- Go back to File Manager, click on **Lists** and then click on ABC.FLIST. In the display of ABC.FLIST, you will now see *all* files matching the wildcard description that you created when you edited the ABC.FLIST. If your files are the same as our example above, you will see files C:\MYPATH\ABC1.TXT, C:\MYPATH\ABC2.TXT and C:\MYPATH\ABC3.TXT in the list, even though there is only one actual entry in the FILELIST, because it is a wildcard entry.

The **AUTOFAV** facility allows files to be automatically added to the Favorite FilesList or to a Named Favorites File List. See [Using AUTOFAV to add to File Lists](#) for more information.

General features of File Lists

- A File List may contain any number of wildcard and normal file name entries, in any combination.
- A File List file can be renamed or deleted if desired.

Creating File Lists with the MAKELIST command

To assist in creating File Lists there is a command, **MAKELIST** to assist. The **MAKELIST** command is only available within File Manager (it makes no sense elsewhere) and is used to create a File List containing the contents of the existing File Manager display.

How you 'create' the list of files within File Manager does not matter. It could be from a File Path/Name and File Pattern request, from a display of another File List, or as the result of a [Find in Files](#) operation. The **MAKELIST** command simply saves whatever list of files is currently displayed as a new File List. e.g. Issuing a **MAKELIST MYLIST1** command would save the list of filenames as **MYLIST1.FLIST**; you can redisplay this list at any time now with a [RECALL](#) command like **RECALL MYLIST1** or **RC MYLIST1**.

The optional operand **SYM** may be added to the MAKELIST command if desired. When specified, MAKELIST will create an **FLIST** file which contains generic path requests for each different file path present in the displayed list of files. For example if the current list of files was:

```
C:\Documents\MyPath1\File1.txt
C:\Documents\MyPath1\File2.txt
C:\Documents\MyPath2\File1.txt
C:\Documents\MyPath2\File2.txt
```

and a **MAKELIST NEWLIST SYM** command was issued, the new File List **NEWLIST.FLIST** would contain:

```
C:\Documents\MyPath1\|*
C:\Documents\MyPath2\|*
```

The Recent Files File List

The **Recent Files** File List contains a list of the most recently edited or browsed files, up to some maximum number. Under the File Manager tab in SPFLite Global Options, a field is provided to specify [No. of files in recent lists](#). A maximum of up to 99 files can be stored in the **Recent Files** File List.

When another file is edited that is not in the list, and the **Recent Files** File List already has the maximum number of files in it, the file that was added to the list the furthest back in the past is dropped off the list. .

When a file already in the **Recent Files** File List is edited again, its file name is moved to the top of the **Recent Files** File List, since it has now been edited *more recently* than it had been before.

The **Recent Files** File List, like all File List files, contains only file names and does not have any date or time information in it. The **Recent Files** File List maintains the most-recent file order by physically storing new file names at the beginning of the list.

Because SPFLite automatically maintains the **Recent Files** File List (unlike other File Lists which are modified based on user commands), it is not recommended to manually edit the **Recent Files** File List, nor to place wildcard entries into it. Such wildcard entries will function for a while but will eventually get dropped, and in some cases this may result in the **Recent Files** File List showing duplicate file names.

The Recent Paths File List

The **Recent Paths** File List contains a list of the most recently referenced path names that have appeared in the **File Path** field of the File Manager.

Note: This entry is considered a "File List" for consistency with the terminology used by other lists, but in fact there are **no files** in this list - only **paths** (fully-qualified directory names). When you click on an entry in the **Recent Paths** File List, it opens up a directory display on that path.

Favorite Files and Named Favorites

The **Favorite Files** File List is the list that contains file names specifically designated by the user as being "favorite". Once a file gets on this list, it stays there until being removed by a Forget line command F. Unlike the **Recent Files** File List, a favorite file list has no maximum size.

A file name gets on the Favorite Files File List from a [FAV](#) primary command in an edit session or by the "Add to favorites" File Manager line command **A** or **ADD**.

It is possible to edit a favorites file list to add or remove file names manually, or to create wildcard entries.

When the **FAV** primary command specifies a *list-name*, the edit file name is stored in a *named favorites* file list with a name of **list-name.FLIST**. These user created **FLISTS** can be displayed in File Manager by selecting the **Lists** entry from the Quick Launch bar. You may then select any of these to see the contents thereof.

You can create a **Named Favorites** File List from a directory display or from another File List display, by using the [MAKELIST](#) command.

The Found Files File List

When the [Find in Files](#) command **FF** is issued, the results of the file search are a list of files. This list is always stored in a list called the Found Files File List. If you select **Found** from the Quick Launch bar, the Found Files list will be displayed. This display can **itself** be used as the basis of a further Find in Files search, the new search results are also stored in that same **Found Files** File List, replacing what was there before.

If it is desired to save the results of one search before doing another one, the **Found Files** File List can be renamed using the **R** line command. When you do this, the newly-named File List will appear under Named Favorites.

The Open Files File List

The **Open Files** File List contains the name of every file currently opened in a file tab. Why would this be important? File tabs are quite useful when the number of files opened is relatively small, such as 9 files or fewer. If you had, say, 50 or 100 files open, it is **possible** for the file tabs can be scrolled left and right, but it's not that easy to do. The **Open Files** File List provides an alternative so that you can scroll up and down in an FM-like list

using the Page Up and Page Down keys to find a file you are interested. If you Select, or mouse-click on a file name, it will not open the same file again (since it's **already** opened) but will simply "jump" to the edit screen where that file is opened.

You may issue a small set of commands against these Open files to perform functions like **CANCEL**, **CLONE**, **DELETE**, **DIR** and **END**. See [Working with File Manager](#) for more details.

See also the command [SWAP - Switch to a Selected File Tab](#), which provides additional ways to jump to a desired file tab.

File Manager ALL command and File Lists

You can apply selected File Manager line commands to a File List, and the commands will be applied, not to the File List itself, but to the files named within the File List. For example, to edit all of the files named in a File List, you would issue the File Manager line command **ALL E** for that File List. See [Working with the File Manager](#) for more information.

File List Cleanup and Forgotten Files

You can use the **Forget** command on all types of File List requests, whether generic path requests or specific filename requests. These 'forget' requests are remembered by adding exception entries to the .FLIST file. This has both a good and bad result.

The good result is that you may now do a [RESET F](#) command while viewing a File List with forgotten entries, and restore them to the view.

Similarly, you may use a [RESET X](#) command to 'undo' previous Exclude request to hide file names.

The bad result is that .FLIST files can slowly fill with 'clutter' entries. They still work fine, they just have more entries in them than might otherwise be needed.

If you select **Lists** from the Quick Launch bar to display your .FLIST files you can request a Normalize function for a File List by entering a **NORM** or **N** next to it's name. This action will:

- Remove any null lines.
- Remove any generic path requests where the actual path no longer exists
- Remove any specific file requests where the file no longer exists **or** the file exists and had been previously forgotten.

Working with File Profiles

Contents of Article

[Introduction](#)
[What about files with no extension ?](#)
[The DEFAULT Profile](#)
[What settings are maintained in each Profile ?](#)
[Automatic Colorization linkage](#)
[Managing changes to a Profile](#)
[Profile display](#)
[Profile locking](#)
[Profile USING](#)
[Copying Profile settings](#)

Introduction

There are many different settings and customization options that can be chosen when editing a file, such as whether to use Tab columns and where they are located, Auto Backup and Auto Save options, Caps mode, the definition of WORD characters, etc. With the different types of files you may edit, you will often need different settings for each file type.

Source files used in programming languages usually use different tab settings, colorization options, and delimiters. Text data files likewise often have differing requirements.

SPFLite provides the ability to save these settings individually for each different file type as described by the file extension, such as .BAT .TXT .CPP etc. SPFLite links a file's extension to its Profile, and will handle the creation and maintenance of Profiles automatically as you work with different file types.

Profile information is stored in the **SPFLite** directory in a folder called **Profiles**. A Profile file has the *name* of the profile, and an extension of **INI**. So, the Profile file of the profile name **TXT** is stored in the file **SPFLite\Profiles\TXT.INI**.

What about files with no extension ?

You may occasionally need to store Profile information for files having no extension at all. One use for this is if you were trying to emulate an environment similar to that on an IBM mainframe, using the Hercules mainframe emulator, and you wanted to use mainframe-like dataset naming conventions for your PC files. (Mainframe data sets do not have extensions.)

This can be handled by using an option available on [the General tab](#) of the Global Options dialog, [Use DIR name as Profile when no File extension](#).

If this option is unchecked, then SPFLite will use the **DEFAULT** profile (see **DEFAULT** description below) for files of this type.

If this option is checked, then the **parent directory** where the file resides will be used as the profile name.

For example, suppose all COBOL source files are in a **COBOL** directory. Under mainframe ISPF, it would be common to see these in a dataset named like **MYUSERID.TEST.COBOl** with 'member names' like **MYCODE**.

If your PC source file was stored in **C:\MyUserId\TEST\COBOL\MyCode**, an SPFLite profile name of **COBOL** would be used for the **MyCode** file, because **COBOL** is the parent directory of **MyCode**.

The **DEFAULT** Profile

There are a wide variety of file types, which are still basically just common text files. (e.g. BAT, INI, CSV etc.) You can certainly have separate Profiles for each of these files, but after a while, you may find you have dozens of profiles, most of them having exactly the same set of options chosen. This is not only wasteful of space, but if you decide to alter one or two Profile options, you are faced with then making that change to a large number of Profiles.

Enter the **DEFAULT** profile.

The reserved profile name **DEFAULT** has the following characteristics:

- It is so important that it is checked for at the beginning of every SPFLite run, and if it does not exist you will be prompted to immediately create one before SPFLite will proceed.
- When an edit of a previously unreferenced file type is requested, and SPFLite cannot find an existing Profile file, it displays the following popup message:



Here you can choose to always use the **DEFAULT.INI** Profile for this new file type, proceed to create a separate new Profile for this file type, or select another existing Profile to use for this new file type.

- The **DEFAULT** profile is also used as the model whenever a new file type Profile is created. That is, the profile named **DEFAULT** becomes the "master default", the starting point for newly-created profiles.
- The **DEFAULT** profile is used as the profile for any files which do not have a file extension, if you have not chosen to "Use the DIR name as the profile". (See the description at the beginning of this section)
- The **DEFAULT** profile is also used as the profile for the File Manager tab. Since the File Manager tab is not used for editing, what purpose does it serve there? It is used for the Find in Files command [FF](#). Since this is essentially a **FIND** command, the Find in Files searches will use the **DEFAULT** setting for the **CASE** profile option from the **DEFAULT** profile. (The **CASE** setting and the **FIND WORDS/CHARS** option can be temporarily changed by issuing this commands on the File Manager primary command line.

What settings are maintained in each Profile ?

The following settings are kept uniquely for each different Profile:

<u>ACTION</u>	Specify automatic SAVE / VSAVE
<u>AUTOBKUP</u>	Whether to create a backup file.
<u>AUTOCAPS</u>	Whether to use AUTOCAPS support or not.
<u>AUTOSAVE</u>	Whether to automatically save a file at END processing time.
<u>BNDS</u>	The left/right column boundaries for the file.

<u>CAPS</u>	Whether CAPS is to be forced on or not.
<u>CASE</u>	Default case handling for FIND/CHANGE literals.
<u>CHANGE</u>	Whether string changes are handled in Data Shift (DS) or Column Shift (CS) mode.
<u>COLS</u>	Whether to display a fixed COLS line at top of screen.
<u>COLLATE</u>	The code page used for the character set collating sequence
<u>EOL</u>	The End of Line delimiter type used. e.g. CR, CRLF, etc.
<u>FOLD</u>	Whether lower-case letters are displayed as if upper-case
<u>HEX</u>	Whether HEX editing mode should be used as a default for the file type.
<u>HILITE</u>	Whether to use automatic colorization support, and FIND / CHANGE highlighting.
<u>LRECL</u>	The logical record length for fixed length record types.
<u>MARK</u>	The current column MARK settings.
<u>MASK</u>	The current model MASK line
<u>MINLEN</u>	The minimum logical record length
<u>PAGE</u>	Whether to use PAGE mode for EOLAUTO or EOLAUTONL files
<u>PRESERVE</u>	Whether to retain trailing blanks on text lines or not.
<u>RECFM</u>	The record format of the file.
<u>SCROLL</u>	The default scroll amount for UP / DOWN / LEFT / RIGHT commands.
<u>SETUNDO</u>	The number of UNDO levels to maintain.
<u>SOURCE</u>	The data encoding used by the data (ANSI, UTF8, EBCDIC, etc.)
<u>START</u>	The default positioning of a file when opened.
<u>STATE</u>	Whether persistent STATE information is retained for this file type
<u>SUBARG</u>	The unique SUBARG value for use during SUBMIT processing.
<u>SUBCMD</u>	The SUBCMD value for use during SUBMIT processing.
<u>TABS</u>	The TABS On/Off setting; the <u>==TABS></u> line value specifies the actual Tab locations.
<u>WORD</u>	The current set of valid WORD characters.
<u>XTABS</u>	The default tab spacing if loading files with embedded Tab characters.

Each of the above settings can be altered in either of two ways:

- Directly with a primary command. The command name in each case is the same as the name in the left column above. (There is no **SCROLL** command, though. To change the default scroll amount, just type into the SCROLL field on the edit screen.)
- By using a pop-up dialog reached by using the **PROFILE EDIT xxx** command, where **xxx** is the desired profile name to be altered.

Note: The BNDS, WORD, MARK and TABS line values are modified by altering the model line that are displayed. These model lines appear when you issue a **PROFILE** primary command, or on demand by using the [BNDS](#), [WORD](#), [MARK](#) and [TABS](#) line commands.

Automatic Colorization linkage

If you utilize [Automatic Colorization Files](#), the name of the colorization control file is derived from the name of the Profile or the name of the **USING** parameter for the Profile (see [Profile Using](#), below).

Managing changes to a Profile

Profiles values can be critical to proper edit processing and it is prudent to protect these settings from being altered inadvertently. Here are some tools to assist you:

Profile display

It is often necessary to display the current Profile. This can be done while editing a file of the type involved, by entering the **PROFILE** command (without any extra operands) and pressing Enter. SPFLite will insert a series of lines into the display as follows:

This display shows you the current values for all profile settings.

Profile locking

If you look at the first =PROF> line above, you will notice the word **UNLOCKED**. This means that any changes you make to profile values will be automatically saved. If the value had shown as **LOCKED**, then no changes to the Profile would be saved while the Profile is **LOCKED**. Any of the displayed values can be altered **for the duration of this edit session** using the various profile-modifying commands, but the changes would not be permanently stored.

A profile can be locked once you have a set of values you wish to keep by issuing the **PROFILE LOCK** command. Similarly, if you want to alter one of a locked Profile's settings permanently, you must issue a **PROFILE UNLOCK** command first, make your change, and then issue a **PROFILE LOCK** command to retain the lock going forward.

Note that if the profile is **LOCKED**, the lock prevents it from being permanently altered. The lock does **not** prevent its settings from being temporarily altered during the course of an edit session. Those alterations simply won't be saved.

The keywords **LOCKED** and **UNLOCKED** can also be spelled as **LOCK** and **UNLOCK**.

Profile USING

You may need to work with different file types that all have a common format. For example a programming language may use different file types for main programs, header files, include files, macro files etc. but may all be of the same format and must be treated the same.

Since SPFLite would ordinarily treat all these different file types as independent Profiles, it would make changing a given profile option a problem, because the change would have to be made to **each** of the different file profiles. This is a time-consuming and error-prone process.

SPFLite allows a Profile **USING** option which allows one Profile to reference another Profile's settings. So how does that help?

Say we have a source language like **C (.C)** which also has associated header files (**.H**). All that needs to happen is to create and customize the **C** Profile as you desire. We can refer to this profile as the 'master' profile. Then, in the Profile for **H** you issue a **PROF USING C** command to refer to the 'master profile' and the **H** Profile will inherit all the settings from the **C** Profile, such as the **C** colorization file if automatic colorization support is active. Now, only changes need to be made to the **C** profile, and all profiles which are **USING** the **C** profile will automatically reference the new settings in **C**.

When you are editing a file which is **USING** another 'master' profile, and the master profile is **UNLOCKED**, then any changes you make will be saved in the master profile, meaning they will apply in the future to all file types which use the master profile.

The **USING** keyword can also be spelled as **USE**.

Copying Profile settings

If you find yourself editing a file type for the first time, you may realize that a number of settings that must be customized for this new type are just like another file type which you have already defined. You can quickly copy all the settings from another profile with the **PROFILE COPY xxxx** command. The other profile's settings will replace the one in the current profile. A **PROFILE COPY** operation is a one-time event.

When you **COPY** another profile, the settings of the two profiles are completely independent of each other; a subsequent modification of one of the profiles has no effect on the other profile. This is different than a **PROFILE USING** situation, where one profile is linked to another.

Profiles and the FF (Find in Files) Command

The FF (Find in Files) command in File Manager will search all displayed filenames for your requested string. Since SPFLite supports a wide variety of file data formats (ASCII, EBCDIC, etc.), the FF command requires a File Profile to exist for all the files it is searching so that it can properly read the data for the search.

However, a displayed list of files may contain other file formats which are never edited by SPFLite. If you are aware of these ahead of time, you can specify specific file types for exemption by the FF command. (See [Options -> File Manager](#))

Many times though, files slip through this process and trigger a pop-up during the FF search when it encounters an unknown file type. You will see the following displayed:



This example was triggered by a **.tta** file type. At this point you can choose one of the 5 options.

- Use the **DEFAULT.INI** file for this and any other **.TTA** files seen subsequently.
- Create a new **TTA.INI** profile for this file type. It will be based on the **DEFAULT.INI** file.

- Select another existing Profile to be USEd for .TTA files.
- Skip this .TTA file, and add it to the FF search exemption list for future searches.
- Cancel the entire FF search at this point.

Working with Excluded Lines

Contents of Article

[Introduction](#)
[Basic concepts from ISPF](#)
[Excluded lines and line-command usage](#)
[Excluded-line placeholder display and the HIDE command](#)
[Persistence of excluded lines and the STATE option](#)
[Methods of excluding lines](#)
[Methods of unexcluding lines](#)
[Primary command options X and NX](#)
[Primary command options MX and DX](#)
[Using SORT X](#)
[The consequences of FIND X DX and CHANGE X DX](#)
[The '-' post-exclude and '+' post-unexclude modifiers](#)
[Primary command examples](#)
[LOCATE and excluded lines](#)

Introduction

Users of IBM ISPF have long had the ability to exclude lines from their edit files. SPFLite builds on this idea and adds many new, powerful features to fully exploit the benefits of line exclusion.

Note: User Lines provide a means to segregate lines into two groups, known as **U lines** and **V lines**, where U lines are a set of lines of particular interest to a user at a given point in time, and "non-User" lines are everything else. The terms **User Line** and **U line** mean the same thing; and, **non-User Line**, **V line** and **ordinary data line**, all mean the same thing.

User lines have a number of similarities with excluded lines, but User lines are not hidden and then revealed the way excluded lines are. If you only need two types of lines to work with, using User Lines may be simpler than using excluded lines. While User Lines are simpler, and share many similarities with excluded lines, there are also fewer supported features with User Lines. Both techniques have their place.

See [Working with User lines](#) for more information.

An *excluded* line is a line that exists, but is not visible. When multiple adjacent lines are excluded, the entire range of lines is visually collapsed and represented as a single line with a placeholder. Excluding lines serves a number of purposes. It gets lines "out of the way" when some lines are more important than others at a given time. It segregates lines into two groups, so that a **FIND** or **CHANGE** could be limited to one of the two types of lines (excluded, or non-excluded). Commands like **DELETE ALL X** or **DELETE ALL NX** can quickly get rid of selected, unwanted lines in a file, and so on.

There isn't a single accepted word to describe reversing the effect of excluding a line or its condition. "Unexcluded" does not appear to be a real English word, but it's often used to describe the line itself. SPFLite also uses the term "show" for the *act of re-displaying a line*. The word "reveal" would also be correct but is seldom used, and "include" is never used.

When discussing whether a line is, or is not, excluded, we use the term *exclusion status* or *exclusion state*. The exclusion status of a line is either excluded or unexcluded. Sometimes the exclusion status is simply called **X** or **NX** for short, after the commands and keywords of the same spelling, and you may see that in some messages.

Basic concepts from ISPF

Users of mainframe IBM ISPF are familiar with the look of an excluded line range, like this screen shot from an actual z/OS ISPF session:

The dashed line is an *excluded-line placeholder*, something that simply indicates where the excluded lines are in the file, and how many of them are there. It also permits a line command to be entered in the sequence area of that placeholder line. Such line commands are normally of the “simple” type and not the “block” type.

Unfortunately, there isn't an agreed-upon word or terminology to describe the "excluded-line placeholder". IBM never came up with an official (and shorter) name or term for it in their manuals. Calling it an "excluded *line*" isn't really correct, since that refers to the *data*, not to this visual display, which could represent *more* than one line of data. So, if we are stuck with the accurate (but wordy) description "excluded-line placeholder", then so are you. Hope you don't mind.

Excluded lines and line-command usage

For example, if you wanted to delete excluded lines 2 and 3, you would not use **DD** or even **D2** on the excluded-line placeholder sequence area, but just **D**. The reason you wouldn't use **D2** is that the entire excluded range represented by the placeholder is treated as *if* one line, for purposes of the line command. If you had said **D2** instead, what would happen is that you would delete lines 2 and 3 as a *single unit*, and *then also* delete line 4 as well.

If you like, you can think of **D2** as deleting 2 *units*, where a *unit* is either a single displayed line, or an excluded-line placeholder (that may itself represent multiple lines). If you think of it this way, you will note that such a *unit* always takes up exactly one line *visually* in the edit display.

What if you had a lot of excluded and unexcluded lines interspersed together and you needed to delete several of them? How would you count all these *units* if you couldn't use the sequence numbers any more for that purpose? For all but the smallest examples, it would probably be too hard to count them, and that's where you would employ a **DD** block instead.

Wait a minute! Didn't we just say *not* to use **DD**? Well, yes and no. You wouldn't use **DD** to delete *just* the lines represented by the placeholder. But, you *can* use a **DD** block where excluded-lines placeholders are one or both of the *ends* of the **DD** block. When you do this, all the lines on the **DD** commands (whether excluded or not) as well as everything inside the **DD** block, are deleted.

If you wanted to shift **all** of the excluded line right by 4 columns, place)4 in the sequence area and press Enter; the block mode shift command)) would not be used.

In SPFLite, you don't have to worry about the dashed lines in the sequence area. These dashes, like line numbers in a normal line, instantly disappear the moment you start typing something into the sequence area.

Excluded-line placeholder display and the HIDE command

The ISPF version of the placeholder display is fine for new users unfamiliar with the concept of using excluded

lines, and it looks good in a help document. Prior versions of SPFLite also used a similar display. But, once you get the hang of it, IBM's format is a little wordy and distracting, especially if you have a lot of them in a file. SPFLite now uses a new, streamlined format that dispenses with all these words. It provides just what is needed, and looks like this:

In some cases, even this brief format is too much. For that, you can use the **HIDE** command to collapse these excluded regions down to just an underline. IBM ISPF users familiar with this may be used to saying **HIDE X**. Since there is nothing else you can hide other than excluded lines, SPFLite does not support **HIDE X** as such, but instead uses **HIDE ON** and **HIDE OFF**. A plain **HIDE** means **HIDE ON**, and does the collapsing action. **HIDE OFF** or **RESET HIDE** will reverse this.

In order for **HIDE** to be most effective, the fixed-width font you use in SPFLite for editing needs to be capable of being underlined in a readable way. *Courier New* meets this requirement, as do fonts in the supplementary SPFLite fixed font library, such as *Raster*.

Example (before **HIDE**):

File Manager EXCLUDED.TXT | Command > HIDE_ | Scroll > HALF

```
***** **** Top of Data ****
000001 Line One
----- < 000002 > -----
000004 Line Four
***** Bottom of Data ****
```

Edit * Lines: 4 Cols 1 to 82 Bnds: MAX OVR T CS S-

After HIDE:

The screenshot shows a window titled 'EXCLUDED.TXT - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED.TXT'. The 'File Manager' tab is selected, and the file 'EXCLUDED.TXT' is highlighted. The 'Command' field contains the command 'EXCLUDED.TXT'. The 'Scroll' field is set to 'HALF'. The main text area displays the following content:

```
***** **** Top of Data ****
000001 Line One
000004 Line Four
***** **** Bottom of Data ****
```

The lines '000001 Line One' and '000004 Line Four' are displayed in red, indicating they are excluded. The status bar at the bottom shows 'Edit *' and other file metadata.

Persistence of excluded lines and the STATE option

What “happens” to excluded lines? Nothing happens to the *data*, it just isn’t displayed, and you don’t lose any data when you save or close the file while lines are excluded. What about the *fact* that a line is excluded? Is that *fact* saved or lost once you close the file? In IBM ISPF, that information is lost. In SPFLite, this information can be retained, it depends on a PROFILE setting called **STATE**. See [Edit STATE saving](#) for a full description of State saving.

Methods of excluding lines

How do lines get excluded in the first place? There are several ways:

- The direct approach: Place **X** on a line, or place an **XX/XX** pair on a block of lines to exclude them. For an **XX** block, if there are any excluded lines already present within it, they will get merged together in one big excluded block. An **XX** line command can be put on a line which already displays the excluded-line placeholder; you would do that to make the excluded region even bigger than it already was.
- The **TX** or **TXX** line command can be used to ‘toggle’ the exclusion state of one or more lines. If the lines were previously unexcluded, they will be unexcluded; if the lines were previously excluded, they will be unexcluded. Lines in the range of a **TX/TXX** line command could be a mixture of excluded and unexcluded lines. Each line is handled individually.
- The **X** line command can take the form of **Xn** where **n** is a number of lines, or it may appear as **X/** or **X** using the new **/** forward and **** backward modifiers.
- The **EXCLUDE (X)** primary command will exclude lines based on a search string, a line range, a tag name, a CC block, or it can exclude all lines.
- The **NEXCLUDE (NX)** primary command will exclude lines similar to **EXCLUDE**, but it works by excluding lines in which a search string is NOT found. For that reason, **NEXCLUDE (NX)** requires a search string to be specified, whereas in **EXCLUDE (X)** you can omit the string if desired.
- **FIND** and **CHANGE** (and related) primary commands can now accept a new keyword option of **MX**, which means *make excluded*. Any lines found in the course of processing a **FIND MX** or **CHANGE MX** will be made excluded.
- The “**–**” modifier on certain line commands can be used to exclude lines after the primary purpose of the line command is completed.
- The **LINE** primary command can be used to apply the **X** or **XX** line command to a range of lines.
- A command of the form **LOCATE condition ALL MX** can be used to exclude all lines having a certain condition, such as **LABEL**, **CHANGED** or **RED**.

Methods of unexcluding lines

Once a given line is excluded, how can it ever get unexcluded? Again, there are many ways:

- **RESET EXCLUDED, RESET X** or plain **RESET** will unexclude all lines (or just **RES** for short)
- The **S** and **SS** line commands will show (unexclude) a range of lines. If an **S** command is placed on an excluded-line placeholder, the entire region is unexcluded. **S** and **SS** are the opposite of **X** and **XX**. The SPFLite **S** line command is a new use for this command name
- The **TX** or **TXX** line command can be used to 'toggle' the exclusion state of one or more lines. If the lines were previously unexcluded, they will be unexcluded; if the lines were previously excluded, they will be unexcluded. Lines in the range of a **TX/TXX** line command could be a mixture of excluded and unexcluded lines. Each line is handled individually.
- The **SHOW** and **NSHOW** primary commands will unexclude selected lines; these are the opposite of **EXCLUDE** and **NEXCLUDE**.
- When a **FIND**, **CHANGE** or similar primary command locates a search string on an excluded line, it will normally unexclude or "pop out" such lines
- The "+" modifier on certain line commands can be used to unexclude lines after the primary purpose of the line command is completed.
- To display just the first 'n' or the last 'n' lines in an excluded region, the **F** and **L** line commands are available.
- IBM ISPF has a seldom-used line command called **S**, which shows the indentation level of a group of excluded lines. This command still exists in SPFLite, but is renamed to **SI** for **Show Indentation**. The old IBM ISPF functionality of **S** is retained for compatibility as **SI**, although experience has shown it to be of limited usefulness.
- The **LINE** primary command can be used to apply the **S** or **SS** line command to a range of lines. Other line commands applied via the **LINE** primary command (except for **X** and **XX**) will generally cause already-excluded lines to pop out and be unexcluded.
- A command of the form **LOCATE condition ALL** can be used to locate and then unexclude all lines having a certain condition, such as **LABEL**, **CHANGED** or **RED**.

Primary command options **X** and **NX**

Traditional ISPF allows for the options **X** or **NX** to be used on certain primary commands. **X** or **NX** on a command is used to decide which lines should be used or acted upon, based on their current exclusion status. A command **FIND ABC ALL X** will find all instances of the string ABC on excluded lines and report the number thereof, and **FIND ABC ALL NX** will only look at non-excluded lines. The **X** and **NX** options are allowed on these commands:

CHANGE
COMPRESS
CREATE
CUT
DELETE
FIND
FLIP
LC
LINE
NFIND

PRINT
REPLACE
SC
SHOW
SORT
SUBMIT
TAG
TC
UC

X and **NX** are not allowed on the following commands. Either the command itself limits the scope of lines considered to only one type, or else it simply doesn't apply:

DROP	<i>X NX not supported</i>
EXCLUDE	<i>only non-excluded lines considered</i>
FLIP	<i>X NX not supported</i>
KEEP	<i>X NX not supported</i>
NEXCLUDE	<i>only non-excluded lines considered</i>
NSHOW	<i>only excluded lines considered</i>
RESET	<i>X NX not supported</i>
SHOW	<i>only excluded lines considered</i>

Primary command options **MX** and **DX**

When a command such as **FIND** or **CHANGE** finds the requested search string on a line, and that line where the search string is found happens to be excluded, the standard action in ISPF is to unexclude that line. When this happens, those excluded lines are sometimes said to "pop out", since visually, that's what it looks like when it happens.

Up until now, this standard action (to unexclude) was the only action available. With SPFLite, there are now two new possible actions that can take place.

Using the **DX** option (*don't change exclusion*), commands such as **FIND** or **CHANGE** will leave the exclusion status of found lines unchanged after the command completes. That means previously excluded lines will remain excluded, and previously unexcluded lines will remain unexcluded, even though the search string had been located.

Using the **MX** option (*make excluded*), when commands such as **FIND** or **CHANGE** locate their search string, will force the lines the string is found on to be excluded after the command completes, regardless of what their prior exclusion status was (excluded or not excluded).

The [LOCATE](#) command will accept an **MX** option if **ALL** is also specified, but it will not accept the **DX** option.

MX and **DX** should not be confused with **X** or **NX**. One way to think of these options is that **X** and **NX** affect which lines are selected *beforehand* to be inspected or changed, but **MX** and **DX** affect what is done with those lines *afterwards*. The distinction is important to understand, because **X** or **NX** can be combined with **MX** or **DX** on the same command.

Example: Changing all occurrences of ABC to DEF only on excluded lines, and then *leaving* them excluded, can be done with the command:

CHANGE ABC DEF ALL X DX

Using **SORT X**

Users of the IBM ISPF have the ability to sort using excluded or non-excluded lines. When the **SORT** command completes, the sorted lines are successively stored back in a way that respects the excluded or unexcluded status of the original lines, in the same relative locations as previously occupied, but in a different order. This can be done even when the excluded or non-excluded lines are in non-contiguous locations.

This action, in which sorted lines are returned to non-contiguous locations, could be called storing those lines in a “scattered” or “interspersed” manner. It’s a little unusual, and won’t often be needed, but it can be useful at times.

In ISPF, when **SORT** is applied to excluded lines, the sorted lines remain excluded.

However, in SPFLite, the act of sorting excluded lines, and then storing them back, is considered a *change*. Since a change to a line, just like with the **CHANGE** command, defaults to unexcluding or “popping out” of excluded lines, this part of SPFLite’s **SORT** behavior is not quite the same as in ISPF.

To get the same results in SPFLite when doing **SORT X** as ISPF produces, it is necessary to add **DX** when sorting these excluded lines. You must say **SORT X DX** to prevent the sorted lines from being unexcluded afterwards.

The consequences of **FIND X DX** and **CHANGE X DX**

Because the new exclusion options **MX** and **DX** are not available in standard ISPF, users may find themselves encountering a situation that never existed before on the mainframe. Recall that the standard action in ISPF is to unexclude an excluded line when found by a **FIND** or **CHANGE**. If you were to do such commands one at a time (repeating F5 or F6, for example), you would see lines “pop out” as they were found or changed, and the cursor would be successively placed on the line where this happened each time.

Now, suppose you have a group of excluded lines, and on each of them there is a string ABC which you are finding, or changing to DEF. And, for some reason, you want the lines *not* to pop out, but be left excluded, so you do this:

FIND ABC X DX

or

CHANGE ABC DEF X DX

That will do the job, alright, but after the first time, or the second time, etc. where is the cursor? Since the lines are *still* excluded, all you have is the excluded-line placeholder, that dashed line with the count of excluded lines. How are you supposed to keep track of the cursor?

SPFLite helps out here by continuing to display the line and column number at the bottom on the status line. When you issue commands like this, SPFLite realizes it has gotten within the “interior” of an excluded range.

This is something that traditional ISPF can’t do, since, as soon as it finds something, it unexcludes it. ISPF can *never* be within the “interior” of a block of excluded lines, but SPFLite *can*.

When the cursor is in the interior of an excluded range, the line and column display is shown in reverse video, to remind you that something special is going on. In addition, you will see the visible cursor placed on the excluded-line placeholder line, in the same relative location it would have had, if the excluded lines had been visible.

Example: Let’s say this is our original file, and we exclude lines 2 and 3. Here, we are going to indent the word LINE to emphasize what’s happening. Notice line 1 starts in column 1; line 2 starts in column 2, etc.

File Manager EXCLUDED.TXT

Command >

```
==COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8-
***** Top of Data *****
```

```
00000001 Line One
x2| Line Two
00000003 Line 3
00000004 Line 4
```

```
***** Bottom of Data *****
```

Edit L 00000002 Lines: 4 Cols 1 to 81 Bnds: MAX INS T DS S- Line Len 0010

Then, find the first occurrence of LINE in the excluded area:

File Manager EXCLUDED.TXT

Command > F line X DX

```
==COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8-
***** Top of Data *****
```

```
00000001 Line One
```

```
----- < 000002 > -----
```

```
00000004 Line 4
```

```
***** Bottom of Data *****
```

Edit 2018-08-09 14:30:11 Lines: 4 Cols 1 to 81 Bnds: MAX INS T DS S- Profile: TXT

That's on line 2 – but we can't see line 2 right now. And because the **FIND** has a **DX** on it, line 2 is going to *stay* excluded. What will the display look like after you do the **FIND** command?

File Manager EXCLUDED.TXT

Command >

```
==COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8-
***** Top of Data *****
```

```
00000001 Line One
```

```
----- < 000002 > -----
```

```
00000004 Line 4
```

```
***** Bottom of Data *****
```

Edit L 00000002 C 2 Lines: 4 Cols 1 to 81 Bnds: MAX OVR T DS S-

Notice the cursor is on column 2 of the placeholder. You will also see the Line/Column display on the status line showing the position of line 2, column 2, in reverse video, like this: **L 000002 C 2**. If you press F5 to repeat the **FIND**, the cursor will advance to line 3, column 3, because that's where the next "LINE" is located:

File Manager EXCLUDED.TXT

Command > Scroll > CSR

```
====COLS> -----+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8---  
***** Top of Data *****  
00000001 Line One  
----- < 000002 > -----  
00000004 Line 4  
***** Bottom of Data *****
```

Line 1: 00000001 Line One

Line 4: 00000004 Line 4

Bottom of Data

Line 1: 00000003 C 3

Line 2: 00000004 Line 4

Line 3: 00000005 Line 5

Line 4: 00000006 Line 6

Line 5: 00000007 Line 7

Line 6: 00000008 Line 8

Line 7: 00000009 Line 9

Line 8: 0000000A Line 10

Line 9: 0000000B Line 11

Line 10: 0000000C Line 12

Line 11: 0000000D Line 13

Line 12: 0000000E Line 14

Line 13: 0000000F Line 15

Line 14: 00000010 Line 16

Line 15: 00000011 Line 17

Line 16: 00000012 Line 18

Line 17: 00000013 Line 19

Line 18: 00000014 Line 20

Line 19: 00000015 Line 21

Line 20: 00000016 Line 22

Line 21: 00000017 Line 23

Line 22: 00000018 Line 24

Line 23: 00000019 Line 25

Line 24: 0000001A Line 26

Line 25: 0000001B Line 27

Line 26: 0000001C Line 28

Line 27: 0000001D Line 29

Line 28: 0000001E Line 30

Line 29: 0000001F Line 31

Line 30: 00000020 Line 32

Line 31: 00000021 Line 33

Line 32: 00000022 Line 34

Line 33: 00000023 Line 35

Line 34: 00000024 Line 36

Line 35: 00000025 Line 37

Line 36: 00000026 Line 38

Line 37: 00000027 Line 39

Line 38: 00000028 Line 40

Line 39: 00000029 Line 41

Line 40: 0000002A Line 42

Line 41: 0000002B Line 43

Line 42: 0000002C Line 44

Line 43: 0000002D Line 45

Line 44: 0000002E Line 46

Line 45: 0000002F Line 47

Line 46: 0000002A Line 48

Line 47: 0000002B Line 49

Line 48: 0000002C Line 50

Line 49: 0000002D Line 51

Line 50: 0000002E Line 52

Line 51: 0000002F Line 53

Line 52: 0000002A Line 54

Line 53: 0000002B Line 55

Line 54: 0000002C Line 56

Line 55: 0000002D Line 57

Line 56: 0000002E Line 58

Line 57: 0000002F Line 59

Line 58: 0000002A Line 60

Line 59: 0000002B Line 61

Line 60: 0000002C Line 62

Line 61: 0000002D Line 63

Line 62: 0000002E Line 64

Line 63: 0000002F Line 65

Line 64: 0000002A Line 66

Line 65: 0000002B Line 67

Line 66: 0000002C Line 68

Line 67: 0000002D Line 69

Line 68: 0000002E Line 70

Line 69: 0000002F Line 71

Line 70: 0000002A Line 72

Line 71: 0000002B Line 73

Line 72: 0000002C Line 74

Line 73: 0000002D Line 75

Line 74: 0000002E Line 76

Line 75: 0000002F Line 77

Line 76: 0000002A Line 78

Line 77: 0000002B Line 79

Line 78: 0000002C Line 80

Line 79: 0000002D Line 81

And the Line/Column display on the status will now show **L00003 C 3**. A final F5 will result in “Bottom of data reached” because there are no more “LINE” strings that are excluded, and the Line/Column display will go back to its normal appearance.

What about **FIND X DX** or **CHANGE X DX** when the excluded lines are *hidden* as a result of the **HIDE** command? You can still do this, but because **HIDE** causes even the excluded-line placeholder to disappear, there is no “interim place” for the cursor to be displayed while you are successively going through the excluded lines, and the editor does not support this situation.

So, SPFLite will only allow you to use your **FIND X DX** or **CHANGE X DX** against *hidden* excluded lines if you specify **ALL** as part of the command. If you can't say **FIND ALL X DX** or **CHANGE ALL X DX** when **HIDE** is in effect, you will have to set **HIDE OFF** first.

The '-' post-exclude and '+' post-unexclude modifiers

If you move or copy one or more lines from one part of the file to another, and any of those lines were originally excluded, they stay excluded afterwards. So, moving and copying lines does not normally change their exclusion status. Suppose you *did* want to change the exclusion status of these lines. Let's say you are 'gathering' blocks of lines from several places in a file and placing the copied lines at some point. As you continue this copying process, the copied lines will take up more and more 'space' in the file, and you will have to scroll through or scroll past them as you continue working. Eventually all those lines will start "getting in your way". It would be convenient if you could tell the editor, 'just exclude the copied lines for right now; I'll get back to them later'. How could you do it?

For sake of discussion, we will assume that you are going to copy lines using **CC/CC** and an **A** command. Suppose you had a file like this, and are copying lines as shown:

When you are done, the file display will look like this:

```
***** * Top of Data *****
000001 Line One
000004 Line Four
000005 // Five
000006 -- Six
000007 Line One
000010 Line Four
***** * Bottom of Data *****
```

Lines 1-4 have been copied and are now lines 7-10. The copies of originally excluded lines 2-3 remain excluded as lines 8-9. If you wanted lines 7-10 “all one way”, you could now go back and manually exclude them with an **XX** line command, or you could manually unexclude them with an **SS** line command, or you could use the primary command equivalents of these. But suppose you didn’t *want* to do that – you just wanted your newly copied lines to *already* be in a certain exclusion state “right off the bat”. Here’s where the “+” and “-” modifiers come into play. Let’s say you want all your copied lines to be excluded right when the copy is done. This is easily done by placing a minus after the **A** command:

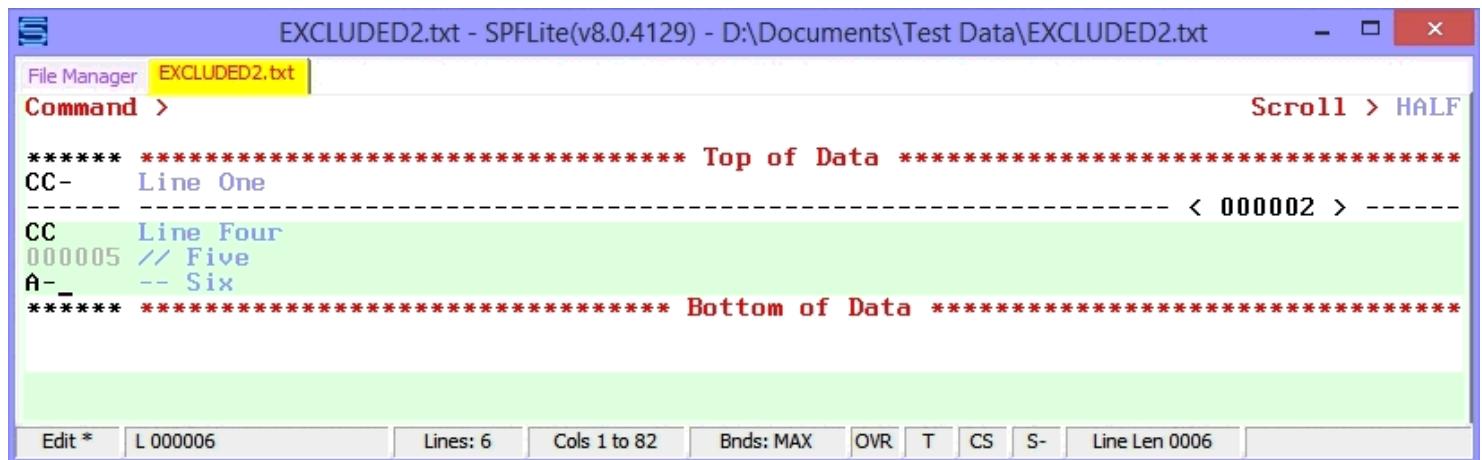
```
***** * Top of Data *****
CC Line One
CC Line Four
000005 // Five
A- -- Six
***** * Bottom of Data *****
```

When you are done, the file display will look like this. Notice that the copied lines are *immediately excluded* for you, so you don’t have to go back and do it yourself manually. You can continue this process until you’ve copied everything you need, and *then* go back and unexclude them later and do further editing when it’s more convenient.

```
***** * Top of Data *****
000001 Line One
000004 Line Four
000005 // Five
000006 -- Six
***** * Bottom of Data *****
```

The minus (or plus) could be on the **CC** as well. What that does is exclude (or unexclude) the block of *source lines* after the copy operation is completed. That happens independently of what happens to the *target lines*.

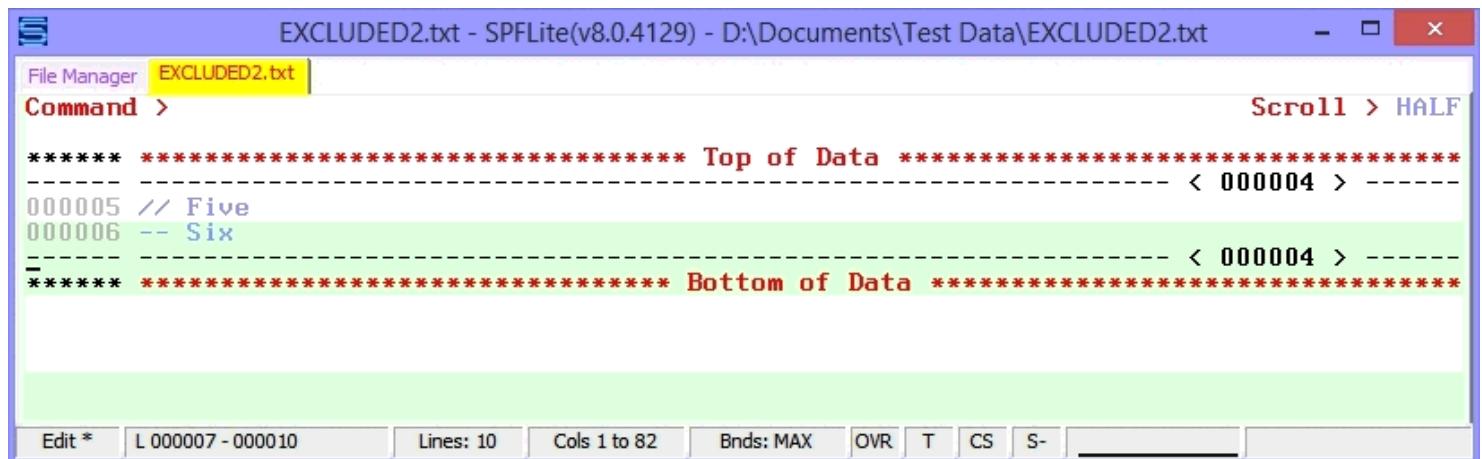
copied after the **A-**.



```
***** **** Top of Data ****
CC- Line One
-----
CC Line Four
000005 // Five
A- -- Six
***** **** Bottom of Data ****

Edit * L 000006 Lines: 6 Cols 1 to 82 Bnds: MAX OVR T CS S- Line Len 0006
```

When you are done, the file display will look like this:



```
***** **** Top of Data ****
< 000004 >
000005 // Five
000006 -- Six
***** **** Bottom of Data ****

Edit * L 000007 - 000010 Lines: 10 Cols 1 to 82 Bnds: MAX OVR T CS S- Line Len 0006
```

Likewise, the **+** can be used to unexclude lines being copied or moved. The **Before** command **B** can accept a **+** or **-** in the same way **A** can.

Like anything else you would do with a block command, if you have some modifier or number on a block, you can put the modifier on one end, or on the other end, or on both ends, but if you use both they must agree. You can't have a **CC+** matched with a **CC-** and not expect SPFLite to complain.

Line commands like **D** and **M** don't take **+** and **-** modifiers, because that would be asking the editor to delete a line and *then* go back and exclude or unexclude a line that no longer exists. (SPFLite is good, but it's not *that* good.) You can use an ordinary **M/MM** block and move it using **A+**, **A-**, **B+** or **B-**, but the **M/MM** can't use the **+** or **-** itself.

Primary command examples

In order to get a feel for how the various primary commands operate under the control of excluded lines, here are a number of examples, with commentary as to what the commands mean and what they would accomplish. These examples will not show every possible feature of every command, but only those that relate to how they interact with excluded lines.

The examples will generally show the use of the **ALL** option. Many of these commands will have other options for selecting lines, such as a line label, a line-label range, line tags and **CC** or **MM** blocks. Line tags and **CC** or **MM** blocks are new features that are explained in another section. For clarity, the full names of commands are shown, like **FIND** and **CHANGE**, whereas in practice people normally use the abbreviations like **F** and **C**, which are shown in parentheses.

LOCATE (L):

LOCATE ALL LABEL

Find all occurrences of labeled lines, and as a side-effect, unexclude all of them. When **ALL** is used on LOCATE, no particular line is located.

LOCATE ALL LABEL MX

Find all occurrences of labeled lines, and as a side-effect, make all of them excluded. When **ALL** is used on LOCATE, no particular line is located.

The LOCATE ALL handling of excluded lines is described [here](#).

FIND (F):

FIND ALL ABC X

Find all occurrences of ABC within excluded lines. Each excluded line having ABC is unexcluded and “pops out” of the display. When ALL is used, a count of strings is displayed. Lines already unexcluded remain unexcluded, but they are not looked at for purposes of determining the count.

FIND ALL ABC X DX

Find all occurrences of ABC within excluded lines. Each excluded line having ABC remains excluded afterwards.

FIND ALL ABC NX

Find all occurrences of ABC within unexcluded lines. Each unexcluded line having ABC remains unexcluded. When ALL is used, a count of strings is displayed. Any excluded lines remain excluded, and they are not looked at for purposes of determining the count.

FIND ALL ABC NX MX

Find all occurrences of ABC within unexcluded lines. Each unexcluded line having ABC will be newly excluded. When ALL is used, a count of strings is displayed. Any formerly excluded lines remain excluded, and they are not looked at for purposes of determining the count.

NFIND (NF):

NFIND ALL ABC X

Find all excluded lines in which ABC is NOT found. Each excluded line NOT having ABC is unexcluded and “pops out” of the display. When ALL is used, a count of *lines* is displayed. (The count is a count of *lines*, not *strings*, because we are finding lines where the string is NOT there. The *lines* exist, but the *strings* don’t, so we can’t very well count something that doesn’t exist!) Lines already unexcluded remain unexcluded, and they are not looked at for purposes of determining the count.

NFIND ALL ABC X DX

Find all excluded lines in which ABC is NOT found. Each excluded line NOT having ABC remains excluded afterwards.

NFIND ALL ABC NX

Find all unexcluded lines in which ABC is NOT found. Each unexcluded line NOT having ABC remains unexcluded. When ALL is used, a count of *lines* is displayed. Any excluded lines remain excluded, but they are not looked at for purposes of determining the count.

NFIND ALL ABC NX MX

Find all unexcluded lines in which ABC is NOT found. Each unexcluded line NOT having ABC will be newly excluded. When ALL is used, a count of *lines* is displayed. Any formerly excluded lines remain excluded, and they are not looked at for purposes of determining the count.

CHANGE (C):

CHANGE ALL ABC DEF X

Find all occurrences of ABC within excluded lines, and change ABC to DEF. Each excluded line having ABC is unexcluded and “pops out” of the display. When ALL is used, a count of strings is displayed. Lines already unexcluded remain unexcluded, but they are not looked at for purposes of determining the count.

CHANGE ALL ABC DEF X DX

Find all occurrences of ABC within excluded lines, and change ABC to DEF. Each excluded line having ABC remains excluded afterwards.

CHANGE ALL ABC DEF NX

Find all occurrences of ABC within unexcluded lines, and change ABC to DEF. Each unexcluded line having ABC remains unexcluded. When ALL is used, a count of strings is displayed. Any excluded lines remain excluded, but they are not looked at for purposes of determining the count.

CHANGE ALL ABC DEF MX

Find all occurrences of ABC within unexcluded lines, and change ABC to DEF. Each unexcluded line having ABC will be newly excluded. When ALL is used, a count of strings is displayed. Any formerly excluded lines remain excluded, but they are not looked at for purposes of determining the count.

EXCLUDE (X):

EXCLUDE ALL

Exclude every line in the file.

EXCLUDE ALL 'ABC' WORD

Exclude all lines having the word 'ABC'

NEXCLUDE (NX): (requires a search string)

NEXCLUDE 'ABC' ALL

Excludes all lines NOT having ABC somewhere on the line. If you begin with no exclusions, like after a RESET is issued, and then issue NX ABC ALL, only those lines containing ABC will be visible, and everything else will be excluded. Some editors refer to this capability as an 'ONLY' function.

Note that for NEXCLUDE, the string operand is required. (Otherwise, you'd be asking SPFLite to exclude all lines in which “nothing” is not present, which doesn't make sense.)

Tritus SPF users will recognize the NX command.

SHOW:

SHOW ALL

Unexcludes all lines in the file or in a CC block. This performs the same function as a RESET command, except that SHOW respects CC blocks.

SHOW 'ABC' WORD ALL

Unexcludes all excluded lines in the file or in a CC block which contain the word 'ABC'.

SHOW 'ABC' PREFIX .A .B

Unexcludes all excluded lines in range of .A .B which contain the prefix 'ABC'.

NSHOW: (requires a search string)

NSHOW 'ABC' WORD ALL

Unexcludes all excluded lines in the file or in a CC block which do NOT contain the word 'ABC'.

NSHOW 'ABC' PREFIX .A.B

Unexcludes all excluded lines in range of .A.B which do NOT contain the prefix 'ABC'.

UC, LC, SC, TC:

These four commands all modify the alphabetic case of one or more lines in some way. Since they are so similar, they are discussed together, using **UC** as an example. These commands change text, so they all will unexclude lines by default.

UC ALL X

Convert all excluded lines in the file or in a CC block to upper case, and then unexclude them.

UC ALL X DX

Convert all excluded lines in the file or in a CC block to upper case, and leave them excluded.

UC ALL NX

Convert all unexcluded lines in the file or in a CC block to upper case, and leave them unexcluded.

UC ALL NX MX

Convert all unexcluded lines in the file or in a CC block to upper case, and then exclude them.

CREATE (CRE):

REPLACE (REP):

CREATE file-name X

CREATE file-name NX

REPLACE file-name X

REPLACE file-name NX

REPLACE will overwrite an existing file, whereas CREATE will not; otherwise these two commands work the same way. (However, REPLACE cannot be used to write over the file you are currently editing.) CREATE and REPLACE can select their source data from among excluded or non-excluded lines using the X or NX option. The source lines may be come from a line-label range, a tag name, or a CC or MM block.

COPY:

The COPY command doesn't take X, NX, MX or DX. However, if you can use the A- or B- line commands so that the copied lines will be excluded afterwards. It is legal to use the A+ or B+ commands instead, but lines copied from an external file are normally inserted as unexcluded, so adding + would not change anything.

CUT X:

CUT NX:

CUT can select its source data from among excluded or non-excluded lines using the X or NX option. The source lines may be come from a line-label range, a tag name, or a CC or MM block.

DELETE (DEL):

ISPF, prior versions of SPFLite, and other SPF-style editors such as Tritus SPF all handled the DELETE primary command somewhat differently.

For some of the operands of DELETE, such as X and NX, the ALL option previously was implied but not allowed by SPFLite, but was required by other editors. For others, ALL was not implied and it had to be present. For a simple DELETE ALL, the command was illegal. SPFLite has simplified these rules as follows.

- In most cases, the **ALL** keyword is optional. That means to delete all excluded lines, you may say either **DELETE ALL X** or just **DELETE X**. Either one is legal, and either one does exactly the same thing. So, long-time users of SPF-style editors that required **DEL ALL X** can keep typing it.
- It is important to remember that **ALL** lines that meet the criteria provided to the **DELETE** command will be deleted, *whether the ALL keyword is used or not*.
- If you wish to delete every line in the file, **DELETE ALL** is now a legal command. Be aware that every line in the entire file **WILL** be deleted, so issue this command with care ! **There is no “are you sure” message to confirm a DELETE ALL**. If you do a **DELETE ALL** by mistake, issue the **UNDO** command immediately, and the file will be restored.
- As a safeguard, **DELETE ALL** cannot be abbreviated as **DEL ALL**. Since most users always type the shortest form of a command to save time, **DEL ALL** with no other operands will be treated as an unintended deletion of the entire file. If you attempt this, the command will be rejected and you will be reminded to fully spell out **DELETE ALL** if that is your intent.
- A plain **DELETE** with no other operands is not permitted.

PRINT X :

PRINT NX:

PRINT can select its source data from among excluded or non-excluded lines using the X or NX option. The source lines may be come from a line-label range, a tag name, or a CC or MM block.

FLIP:

The FLIP primary command will invert the exclusion status of a specified range of selected lines or of all the lines in the file. Excluded will become unexcluded, and vice-versa. If the range is omitted, the entire file is “flipped”. What is notable about SPFLite is that FLIP will take a CC block to define its range, in addition to a label range.

RESET (RES):

RESET EXCLUDED will unexclude all excluded line. Since this is the default action, everyone just says RES instead of spelling out the whole command.

SORT: (see also the section [Using SORT X](#) for more information)

SORT X

Excluded lines only are sorted, and returned as unexcluded lines to the same set of line locations, except in sorted order.

SORT X DX

Excluded lines only are sorted, and returned as excluded lines to the same set of line locations, except in sorted order.

SORT NX

Unexcluded lines only are sorted, and returned as unexcluded lines to the same set of line locations, except in sorted order.

SORT NX MX

Unexcluded lines only are sorted, and returned as excluded lines to the same set of line locations, except in sorted order.

RESET (RES):

ISPF provides **RESET** to reset a number of “special” line conditions, such as changed-line markers, profile lines, and excluded lines. If you want to clear your line exclusions, most people just type **RES** on the command line. Doing so will not reset any labels that might be present. In SPFLite, RESET works basically the same way, but

since you now have the possibility of labels, tags and kept line commands, when you really need to reset a number of kinds of special lines, you would ordinarily need several **RESET** commands. A regular **RESET** without any options does not reset labels, tags or kept commands.

To allow everything to be cleared at one time, SPFLite provides **RESET ALL**. This command will reset everything that is possible to reset, including excluded lines, labels, tags, kept line commands, lines with special indicators like ==CHG> and the lines displayed in response to a **PROFILE** command.

Working with User lines

Contents of Article

[Introduction](#)

[Why use User Lines ?](#)

[Primary command usage of User Lines](#)

[Setting the User Line status is repeatable, and what that means](#)

[Comparison of features: Excluded Lines vs. User Lines](#)

[Special handling of RESET when resetting User Line status](#)

[Special handling of co-located label, tag and User state on same line](#)

[Examples](#)

Introduction

Conceptually, all data lines exist in one of two states: either they are "**U lines**" or they are "**V lines**", where **U lines** are simply a set of one or more lines of special interest to the user at some given time, and **V lines** are everything else.

Essentially, the U/V state of a line is like a **bit**: It's **on** if it's a U line, and **off** if it's **not** a U line.

Not being a U line is the same as **being a V line**.

Note: While "**U Line**" is short for "**User line**", what does "**V line**" stand for? To be honest, not much. The **V** doesn't have any especially clever meaning, other than simply "**not being a U line**". The letter "**V**" was chosen for practical reasons, because the line command **V** wasn't being used for anything else at the time. We tried to tie-in **V** with the "**V**" in the primary command names **REVERT** and **NREVERT**, which have abbreviations of **VV** and **NV** respectively. That's about as clever as it gets.

A file will ordinarily consist entirely of **V lines**, which is the default state of a file when no lines have been marked as U lines. You will see the expression "**ordinary V lines**" throughout this Help documentation, to describe lines that are not marked as U lines. Other than being in one or the other of these U/V states, there is no difference between U and V lines and the data lines you have always worked with.

When a data line becomes a U line, a **|** vertical bar character appears in the "gap column" just after the line number. This vertical bar marks the line as being a U line.

The U/V state of a file is persistent if **STATE ON** is in effect. If **STATE OFF** is in effect for a file you're working on, and you set any lines in the file to be U lines, they will all revert back to V lines once the file is closed and reopened.

Note: Because the U/V state of a file affects the **state** but not the **data**, just marking and unmarking User Lines does not count as a "change" to the file, and so during an edit, you will not see the modified-marker like **Edit *** in the lower-left part of the screen. See [Saving the Edit STATE](#) for more information.

Why use User Lines ?

Using U lines can make it convenient to perform extended editing tasks that repeatedly target the same given area of interest within a file, in a way that can be easier than using labels, tags or line exclusion.

As seen below, there is a fairly close comparison between User Lines and excluded lines. If you wanted, you

could also compare User Lines to tagged lines. Suppose you used a tag of :**U** for user lines; much of what User Lines offers could also be done with tags. However, there are drawbacks to both these techniques.

Drawbacks of excluded lines

- When lines are excluded, you can't see them. That can be a good thing at times, but not always.
- **FIND** and **CHANGE** make excluded lines "pop out". If you don't like that, there are the **MX** and **DX** keywords, but that adds complexity, and not all commands support **MX** and **DX**, either.
- If an excluded line "pops out" it's not excluded any more, and so if you used excluded/unexcluded to segregate lines into two groups, lines that "popped out" are now "in the other group". Again, that can be a good thing at times, but not always.
- There are **SORT** issues to consider

Drawbacks of tagged lines

- Tagged lines are powerful, but more complex than excluded lines.
- The **TAG** command contains many options, and may be intimidating for some.
- When a tag is present in a line (as with labels) it obscures the line number sequence field. If you have many tags on many lines, this could be a bit distracting.
- The tag notation is a little lengthy and hard to type, requiring a : colon as a shifted key to enter it.

Does this mean that User Lines are the answer to everything? **No**. Excluded and tagged lines are still needed:

- Excluding lines get lines of lesser interest "out of the way". Often times, that **is** a good thing.
- The "pop out" action in **FIND** and **CHANGE** can reveal strings of special interest; that can be very important when you're trying to find something and you don't know if, or where, it might be.
- Some complex "data mining" actions using **TAG** and **ASSERT** can't be easily done any other way.

The User line status is independent of other line characteristics. Because of that, User Lines don't **have to** be the answer to everything. You can **combine** User Lines and other techniques, for even **more powerful** methods of referencing lines. That means that **any** of these attributes may be applied to a line **independently** of each other:

- a line can be excluded, or not excluded
- a line can have a label, or not
- a line can have a tag, or not
- a line can be designated as a User line, or not

For example, you may set a group of lines as User Lines, and work on them for a while. Then, when you're done with a particular aspect of your work, you can exclude those User Lines and "get them out of the way", but **they are still User Lines**, and **will remain so** if you unexclude them later with a **RESET**, **SHOW** or other command.

As with all tools, you have to pick the right one for the right job.

Primary command usage of User Lines

The **ULINE**, **REVERT**, **NULINE** and **NREVERT** commands allow the "U/V state" of a line to be set, based on the presence or absence of a search string. **Reverting** a line means to revert its U/V state back to **V**, the default state of ordinary data lines.

These four commands may be thought of as follows:

- **ULINE** is like **FIND** or **EXCLUDE**
- **REVERT** is like **SHOW**
- **NULINE** is like **NFIND** or **NEXCLUDE**
- **NREVERT** is like **NSHOW**

Using **U** and **NU** is very similar to operating on lines that either are, or are not, excluded, using the keywords **X** or **NX**. User lines can be combined with line exclusion, line labels and line tags, if desired. During the design phase, we patterned the **U|NU** usage on the existing **X|NX** usage that ISPF and SPFLite users are familiar with. If you'd like an easy-to-remember rule, then just remember that essentially any primary command that takes **X** or **NX** will take **U** or **NU**. The exceptions to this rule are as follows:

- the commands **EXCLUDE**, **SHOW**, **NEXCLUDE** and **NSHOW** don't allow **X|NX** on the command line but **do** allow **U|NU**
- the commands **ULINE**, **REVERT**, **NULINE** and **NREVERT** don't allow **U|NU** on the command line but **do** allow **X|NX**

Why didn't we make the primary command keywords **U** and **V**, instead of **U** and **NU**, since we just got through saying there are **U** and **V** lines, **and** there are **U|UU** and **V|VV** line commands? That was possible, and we did discuss it, but it wouldn't have followed the **X|NX** pattern. So, we have the keywords as **U** and **NU** because it is the more consistent approach.

It's sort of the same reason why we stuck with **X** and **NX**, even though there is a **SHOW** primary command and an **S/SS** line command. Why not introduce an **S** keyword instead of continuing to use **NX**? Because that's not how ISPF does it. Same here. We are trying to follow the "spirit" of how ISPF would have done **this**, if IBM had designed it.

The syntax of the commands **ULINE**, **REVERT**, **NULINE** and **NREVERT** were patterned after the **FIND** command. This includes the string search argument, **CHAR/WORD/PREFIX/SUFFIX** options, color keywords, **X|NX** and **MX|DX**. If you are modifying several different sections of your file to be U lines (or reverting them back to V lines), you can exclude them afterwards by using the **MX** keyword, or keep them excluded if they already are, by using the **DX** keyword. This can help you organize your work, by getting lines you just marked as U lines "out of the way," so that you can concentrate on other parts of your file. Once you have set all the desired lines, you can unexclude them and begin working on them. See [Special handling of RESET when resetting User Line status](#) below, and [Working with Excluded Lines](#) for more information.

In addition, primary commands can be restricted to operating on lines that either are, or are not, U lines, using the keywords **U** or **NU**.

The commands where **U** and **NU** can be specified are as follows:

APPEND, **CHANGE (C)**, **COMPRESS (CP)**, **CREATE (CRE)**, **CUT**, **DELETE (DEL)**, **EXCLUDE (X)**, **FIND (F, FF)**, **FLIP**, **JOIN**, **LC**, **LINE**, **LOCATE (LOC, L)**, **NDELETE (NDEL)**, **NFIND (NF)**, **NEXCLUDE (NX)**, **NFLIP**, **NSHOW**, **PREPEND**, **PRINT**, **PTYPE (PT)**, **REPLACE (REP)**, **SC**, **SHOW**, **SORT**, **SPLIT**, **SUBMIT (SUB)**, **TAG**, **TC** and **UC**.

See also the [U|UU](#), [V|VV](#) and [TU/TUU](#) line commands for more information.

Setting the User Line status is repeatable, and what that means

If you use a command like **ULINE (UU)** or **REVERT (VV)** over a range of lines, that command is repeatable. That is, if you were to retrieve the command (probably with F12) and run the same command again, it would re-mark, or re-unmark, the same set of lines the same way. That is because the same set of conditions (like search strings) would still hold, and unlike the **EXCLUDE** and **SHOW** primary commands, **ULINE** and **REVERT** don't change the visibility of those lines. (Recall that these commands are like **FIND**, which doesn't change the data.)

This property, where you can do the same thing over again, has the peculiar mathematical name of "idempotent", a word which signifies an operation that produces the same results no matter how many additional times you perform it. It is like setting a bit on using in **OR** operator; if you set on a bit that was already on, it will still have the same value it had before.

In some ways, line exclusion is also repeatable, but not to the extent that User Lines are. For example, you can issue a primary command of **EXCLUDE .A.B ALL**, and if you re-issue it, the same line range will be excluded (again). This corresponds to a command like **ULINE .A.B ALL**, which can also be repeated. However, suppose you issue a line command like **X3** on line 1; it will exclude lines 1 to 3. If you issue **X3** again where line 1 used to be, it now has a placeholder representing 3 lines, so putting **X3** there will (again) exclude lines 1 to 3 **as a unit** and then also exclude lines 4 and 5. So, you get more lines excluded the second time, and thus the action is not repeatable. In contrast, if you put a **U3** command on line 1, no matter how many times you do it, it will still only mark lines 1 to 3 as User Lines.

Why bring up this bit of trivia? Recall the beginning of this article, where we said that the **U/V status is like a bit?** If it is, and if setting the "bit" using **ULINE** is like OR-ing the bit, then you could set various ranges of lines using any combination of **ULINE**, **NULINE** and **UU** line commands, and when you were done, the set of lines marked as U lines would be in an **OR relationship**.

For example, assuming the file had no existing U lines, if you did this:

```
ULINE ABC ALL
NULINE DEF ALL
line command U\ on line 10
```

where would U lines be found? They would be:

- all line containing **ABC**
- **OR** all lines **not** containing **DEF**
- **OR** the first 10 lines of the file

Once you have described this **OR relationship**, you can use the **U** or **NU** keywords to work with those lines from the primary command line. For example, suppose you now want to convert all those lines to upper case:

```
UC U ALL
```

Or, suppose you want only those lines written to a temporary file:

```
CREATE TEMP.TXT .ZFIRST .ZLAST U
```

Remember, essentially any command that takes **X** or **NX** will also take **U** or **NU**. So, you can use the U or non-U status to selectively act on a set of (possibly non-adjacent) lines.

By the way, if that **CREATE** command seems a little long, it's because **U** doesn't imply a line range by itself, so we have to give it one. If you do this frequently, you may want to create a SET symbol to shorten it:

```
SET ALL = ".ZFIRST .ZLAST"
```

Then the command becomes (also using the short form of the command itself):

```
CRE TEMP.TXT =ALL U
```

If it turns out that dealing with "all U lines" is something you do a lot, you could make this even shorter:

```
SET U = ".ZFIRST .ZLAST U"
```

and then

```
CRE TEMP.TXT =U
```

Creating SET symbols can be useful, but it's best to limit them to things you do all the time, to avoid creating so many you start forgetting what you have defined or how to use them.

Comparison of features: Excluded Lines vs. User Lines

Feature of Excluded Lines	Feature of User Lines
Primary command EXCLUDE (X)	Primary command ULINE (UU)
Primary command NEXCLUDE (NX)	Primary command NULINE (NU)
Primary command SHOW	Primary command REVERT (VV)
Primary command NSHOW	Primary command NREVERT (NV)
Primary command HIDE	No corresponding primary command for User Lines
Primary command option X	Primary command option U
Primary command option NX	Primary command option NU
Primary command options MX and DX	No corresponding primary command options for User Lines
Line command X	Line command U
Block-mode line command XX	Block-mode line command UU
Line command S	Line command V
Block-mode line command SS	Block-mode line command VV
Line command TX	Line command TU
Block-mode line command TXX	Block-mode line command TUU
LOCATE ALL <i>type</i> will unexclude lines	LOCATE ALL U will unexclude User Lines. There is presently no corresponding LOCATE option to find lines of a certain type and then alter their U/V state.
LOCATE ALL <i>type MX</i> will exclude lines	LOCATE ALL U MX will exclude User Lines. There is presently no corresponding LOCATE option to find lines of a certain type and then alter their U/V state.
Line command F shows first n excluded lines	Line command of Vn is comparable to Fn , provided that the next n lines are not excluded
Line command L shows last n excluded lines	No corresponding line command
Line command SI shows indentation level	No corresponding line command
Post-exclude modifier –	No corresponding line command option for User Lines
Post-unexclude modifier +	No corresponding line command option for User Lines
STATE ON retains excluded lines	STATE ON retains User Lines
HIDE conceals excluded lines	No corresponding feature, unless also excluded

Excluded lines disappear	User Lines remain visible unless also excluded
Excluded lines represented by a "placeholder" line	User Lines represented by a vertical mark in gap column
Excluded areas vary in size; affect line commands	No corresponding issues
FIND and CHANGE will make excluded lines appear	FIND and CHANGE do not alter User Line status
FIND X DX can position cursor on hidden lines	No corresponding issue for User Lines
CHANGE X DX can position cursor on hidden lines	No corresponding issue for User Lines
SORT X requires DX for ISPF compatibility	No corresponding issue for User Lines; SORT U allowed
Plain RESET implies RESET X	Plain RESET implies RESET U but only if enabled (see below)

Special handling of **RESET** when resetting User Line status

Long-time ISPF users are accustomed to **RESET** performing a number of default reset-actions when no options are specified. A plain **RESET** is commonly used to reset excluded lines, as if **RESET X** were specified. These default actions can be extended to the new User Line feature. That is, a plain **RESET** can **also** revert any User Lines back to ordinary non-User Lines, as if **RESET U** were specified.

However, since User Lines are a new editing concept, there is no prior experience to form a consensus about how this should best be handled. Some people may want User Lines implicitly reverted, and others might not.

To address this, a plain **RESET** will revert any existing User Lines back to ordinary non-User Lines, but **only** if you enable the checkbox on the **General Options** dialog that says, "**Default RESET will revert user line status**". As with other settings on this dialog, it is global and applies to all SPFLite edit sessions.

This checkbox is initially disabled.

Special handling of co-located label, tag and User state on same line

When a label and a tag exist on the same line, the "gap column" will display a : colon to remind you of this situation, and there is also a special status line display whenever the cursor is on that line. See [Line label and line tag co-location](#) for more information.

It is possible for such a label/tag co-location line to **also** be a User Line. When this occurs, the : colon in the gap column is replaced by the | vertical bar character. The special notation on the status line remains as in previous versions, displaying the tag name for that line.

If you revert this line back to a non-User line, using **REVERT**, **NREVERT**, **RESET U** or the **V/V** line commands, and make no other changes to this line, the : colon in the gap column will reappear.

Examples

Let's start with a simple test file. Notice there is a missing right parenthesis on line 2.

```
***** * Top of Data *****  
000001 (one)  
000002 (two)  
000003 (six)  
000004  
000005 (ONE)  
000006 (SIX)  
000007 (TWO)  
***** * Bottom of Data *****
```

The goal will be to mark the first three lines as User Lines. There are a number of ways this could be done:

- **U3** line command on line 1
- **U** line command on line 3
- **UU .1 .3** primary command

But suppose we are looking for some particular condition to mark. In this case, 'words' of three letters that start with a left parenthesis, where the words are in lower case. We can do this with a **UU** command using a Picture.

(Note that parentheses are not special Picture characters, and don't need to be escaped. If you needed to use some special characters, you can always put a \ backslash in front of them, to have them treated as ordinary data.)

```
***** * Top of Data *****  
000001 (one)  
000002 (two)  
000003 (six)  
000004  
000005 (ONE)  
000006 (SIX)  
000007 (TWO)  
***** * Bottom of Data *****
```

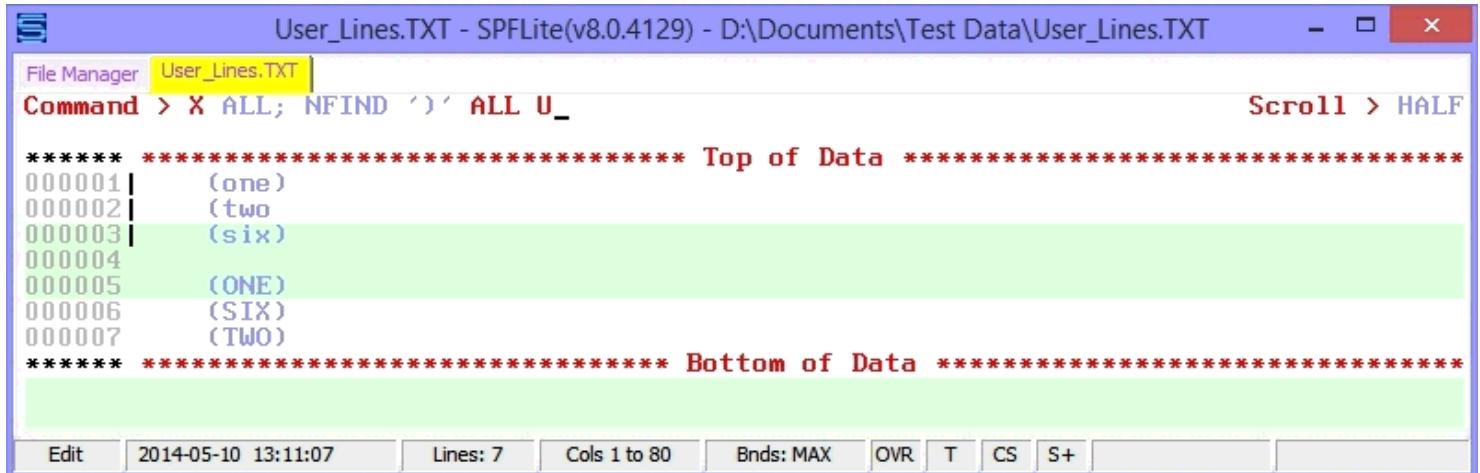
This will mark the desired lines, and then report on the number of lines affected:

Now, noting that 3 lines were marked, suppose we knew (or suspected) that there might be some unmatched parentheses somewhere in the file. We could see if that condition existed in the marked User Lines by doing a **FIND** with a **U** keyword:

The **U** keyword directs the **FIND** command to only look at lines marked as User Lines, which in this case are lines 1-3, as seen by the | vertical bar on those lines. Result:

The **FIND** message confirms that the unmatched parentheses are somewhere in the User Line area. If you wanted to successively look at every U line that had a left parenthesis, to visually inspect which didn't have a right parenthesis, you could issue a **FIND** '(' **U** and then repeat that manually with **F5**.

But, suppose you're in a hurry, and don't **want** to do all that manual inspection. You could exclude all the lines, and then find all User Lines which did not have a right parenthesis:



User_Lines.TXT - SPFLite(v8.0.4129) - D:\Documents\Test Data\User_Lines.TXT

File Manager User_Lines.TXT

Command > X ALL; NFIND ')' ALL U_

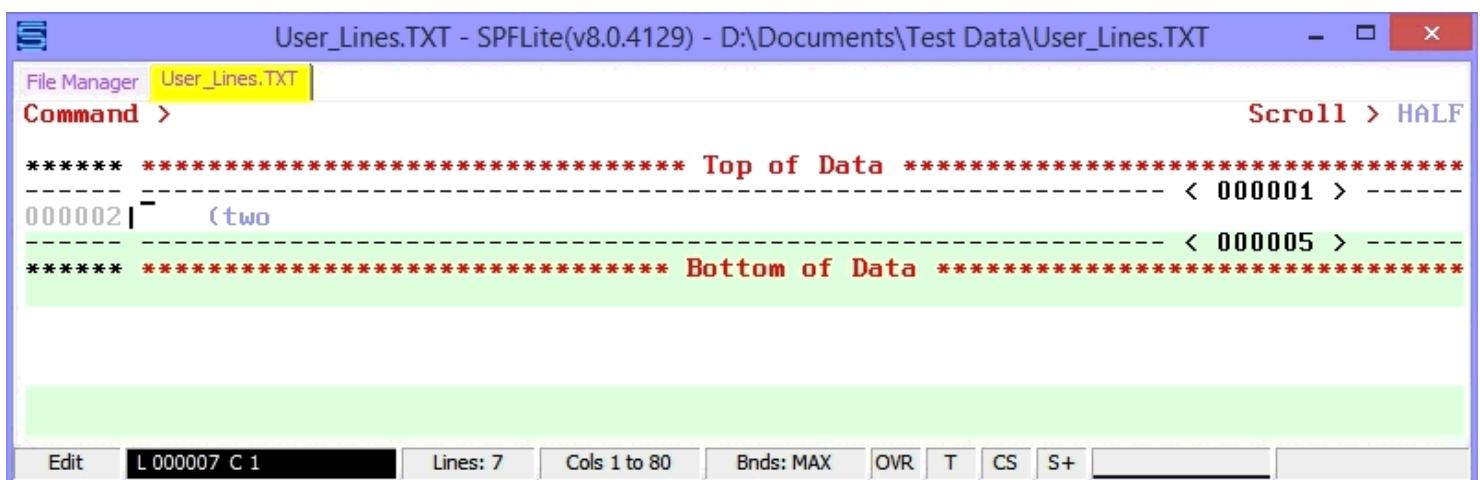
***** * ***** Top of Data *****

000001 | (one)
000002 | (two
000003 | (six)
000004 |
000005 | (ONE)
000006 | (SIX)
000007 | (TWO)

***** * ***** Bottom of Data *****

Edit 2014-05-10 13:11:07 Lines: 7 Cols 1 to 80 Bnds: MAX OVR T CS S+ []

Sure enough, once you do that, the only line left displayed is the one with the missing right parenthesis on line 2:



User_Lines.TXT - SPFLite(v8.0.4129) - D:\Documents\Test Data\User_Lines.TXT

File Manager User_Lines.TXT

Command >

***** * ***** Top of Data ***** < 000001 > -----

000002 | (two ----- < 000005 > -----

***** * ***** Bottom of Data *****

Edit L 000007 C 1 Lines: 7 Cols 1 to 80 Bnds: MAX OVR T CS S+ []

What's nice about User Lines is that, since they act independently of line exclusion, you can **RESET** the file and unexclude (or re-exclude) everything, and perform these tests all over again, or run other ones. When you do, all of your User Line marks remain undisturbed.

Remember that a plain **RESET** command either **will**, or **will not**, clear the User Line marks from your file, depending on the General Options **RESET** checkbox described [above](#).

Working with Multi-Edit Sessions

Contents of Article

[Introduction](#)

[Selecting a set of files for the multi-edit session](#)

[Working with SPFLite's representation of multiple files in a single edit tab](#)

[Storing data during SAVE and END processing](#)

[Simulating a multi-browse with SAVEAS](#)

[Adding and removing files from a multi-edit session](#)

[Line commands permitted on =FILE> lines](#)

[The \(ClipName\) and \(ClipPath\) keyboard primitive functions in Multi-Edit](#)

[Creating and using orphaned lines](#)

[Navigating from one file to another](#)

[Dealing with external changes while an edit is in progress](#)

[Treatment of unqualified file names on primary commands](#)

[Read Only files and MEDIT](#)

[What else can you do in a multi-edit session ?](#)

Introduction

Multi-Edit is a new editing technology, not present in IBM ISPF. A multi-edit is an edit session containing multiple files under a single SPFLite edit tab.

Why use Multi-Edit sessions?

The main reason is increased productivity in quickly making large-scale, synchronized changes to multiple files at one time.

Suppose you have several text files that are interrelated, containing common names and values that are defined and referenced throughout every file. (Program source files, containing the names of functions and variables, are one example of this.) Now, what if you wanted to make a new version of these files that involved changing these common names and values everywhere in every file. How would you go about doing it?

You might have an existing list of files you need to work on, in a directory list or File List. Or, you could use a Find in Files command **FF** to search for all files containing some string, which produces a Found Files File List. Even with the Find in Files command **FF**, now part of SPFLite, you would still have to manually apply the changes to each file, one file at a time. When there are many changes and many files involved, that could be a long, tedious process.

Let's say you wanted to change the word ABC to DEF in every file: You would have open "file 1" from your list, issue a command like **CHANGE ABC DEF WORD ALL**, close the file, open "file 2", and repeat the process, over and over again. Now, picture *lots* of changes to *lots* of files. You can see how labor-intensive this could get.

Ideally, you would like to be able to say, **CHANGE ABC DEF WORD ALL** and have that change applied to every file containing the word ABC, all at the same time. With multi-edit, now you can!

Some users of Windows GUI-type editors and software developers that use Integrated Development Environments (IDE's) are familiar with the powerful concept of a "refactoring editor". Our Multi-Edit facility is not exactly a refactoring editor, but it's pretty close. One advantage of SPFLite's Multi-Edit is that it's a general facility, one that can be used on any data, and is not dependent on a software development environment or any programming language. And, it fits within an ISPF-like architecture, something ISPF itself doesn't offer.

Bear in mind that you could have several multi-edit sessions open at the same time, as long as no file is involved

in more than one edit session (whether a regular edit session or multi-edit session).

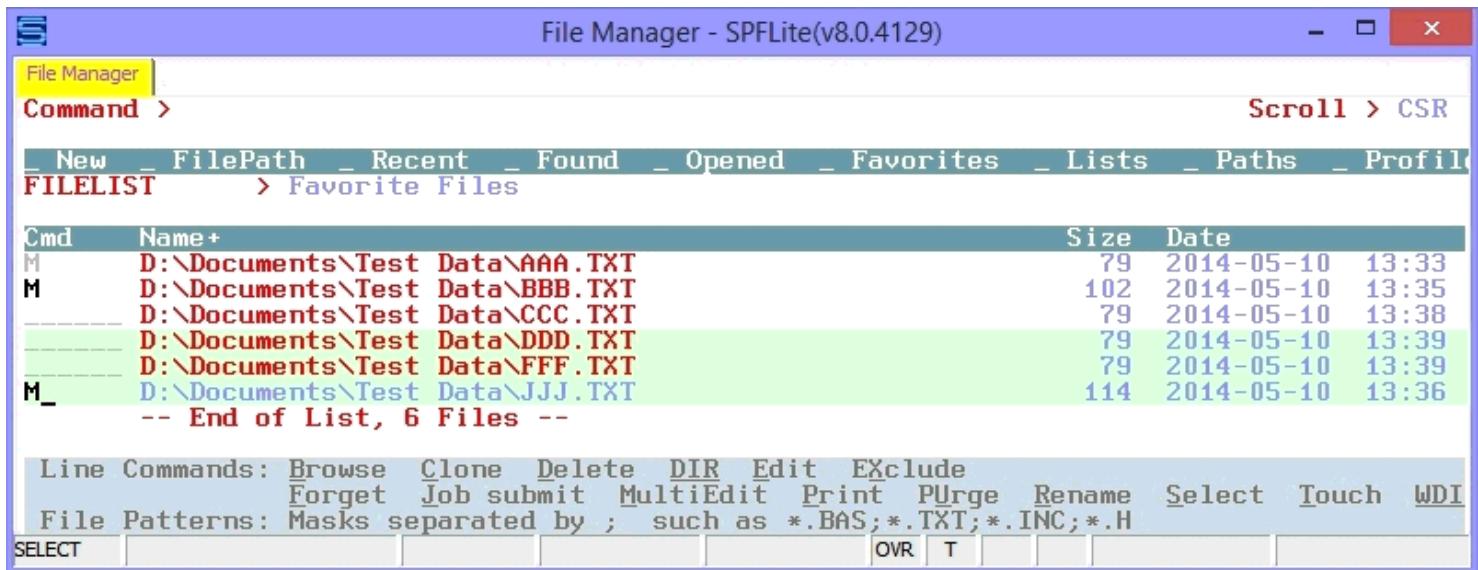
We'll take this one step at a time.

Selecting a set of files for the Multi-Edit session

In the File Manager, start with a displayed list of files, and for each one that you want to be part of your Multi-Edit session, place an **M** line command on that line. The list of files can be a directory list, where you set the File Path and File Patterns fields, or it can be a File List display.

Note: You can apply selected File Manager line commands to a File List, and the commands will be applied, not to the File List **itself**, but to all the files named **within** the File List. In particular, to start a Multi-Edit all of the files named in a File List, you can issue the File Manager line command **ALL M** for that File List. See [File Manager ALL command and FILELISTS](#) for more information.

Let's say you have placed the files of interest into the **Favorite Files** File List. Here is what the list might look like (the display is a little simplified here). We are going to select three files from this list, the AAA, BBB and JJJ files. To do this, place the Multi-Edit line command **M** on each of these files, and after all of the **M** codes are in place, then press Enter. If you have to, you can scroll up and down in the list, but don't press Enter until you are done putting **M** codes on every file you want to edit.



The screenshot shows the SPFLite File Manager interface. The title bar reads "File Manager - SPFLite(v8.0.4129)". The menu bar has "File Manager" and "Command >". The toolbar includes "Scroll > CSR". The main window shows a list of files in a "FILELIST" under the "Favorite Files" tab. The list includes files AAA.TXT, BBB.TXT, CCC.TXT, DDD.TXT, FFF.TXT, and JJJ.TXT. The file JJJ.TXT is highlighted with a green background. The "Cmd" column shows "M" for files AAA, BBB, and JJJ, indicating they are selected for multi-edit. The "Name" column shows the full path and file name. The "Size" and "Date" columns show the file size and last modified date. The bottom of the list shows "-- End of List, 6 Files --". The status bar at the bottom shows "Line Commands: Browse Clone Delete DIR Edit Exclude Forget Job submit MultiEdit Print PUrgo Rename Select Touch WDI" and "File Patterns: Masks separated by ; such as *.BAS;*.TXT;*.INC;*.H".

Cmd	Name	Size	Date
M	D:\Documents\Test Data\AAA.TXT	79	2014-05-10 13:33
M	D:\Documents\Test Data\BBB.TXT	102	2014-05-10 13:35
---	D:\Documents\Test Data\CCC.TXT	79	2014-05-10 13:38
---	D:\Documents\Test Data\DDD.TXT	79	2014-05-10 13:39
---	D:\Documents\Test Data\FFF.TXT	79	2014-05-10 13:39
M	D:\Documents\Test Data\JJJ.TXT	114	2014-05-10 13:36
-- End of List, 6 Files --			

Once you press Enter, you are about to begin your first multi-edit session. (This is something you've never seen before in an ISPF-style editor, and probably not anywhere else, either.)

Working with SPFLite's representation of multiple files in a single Edit tab

Once you have selected your files with the **M** command and pressed Enter, you will have a new file tab opened. Since there are multiple files involved, and no single name would be appropriate for the file tab label, you will instead see **(Multi-Edit)** in the Windows title bar, and **(M-Edit)** in the file tab label. The edit session display will look like this:

File Manager (M-Edit) Command > _ Scroll > HALF

```
***** **** Top of Data ****
=FILE> D:\Documents\Test Data\AAA.TXT
000001 LINE ONE OF FILE AAA.TXT
000002 LINE TWO OF FILE AAA.TXT
000003 LINE THRE OF FILE AAA.TXT
=FILE> D:\Documents\Test Data\BBB.TXT
000004 LINE 1 OF ANOTHER FILE - BBB.TXT
000005 LINE 2 OF ANOTHER FILE - BBB.TXT
000006 LINE 3 OF ANOTHER FILE - BBB.TXT
=FILE> D:\Documents\Test Data\JJJ.TXT
000007 LINE 1 OF YET ANOTHER FILE - JJJ.TXT
000008 LINE 2 OF YET ANOTHER FILE - JJJ.TXT
000009 LINE 3 OF YET ANOTHER FILE - JJJ.TXT
***** **** Bottom of Data ****
```

3 Edit 2014-05-10 13:59:14 Lines: 9 Cols 1 to 80 Bnds: MAX OVR T CS S+

Each of the files in the Multi-Edit session are separated by a **=FILE>** separator line. This separator line cannot be moved, modified or excluded. As you will notice the **=FILE>** separator does not have a line number. This is because it is not 'data' but merely denotes the start of a new file. You will also see that each **=FILE>** line contains the fully-qualified name of the file that follows it. The actual characters **=FILE>** are called the **=FILE>** marker.

Take note of the line numbers. You will see that even though there are three files in this multi-edit session, the line numbers are consecutive from 1 up, but they represent three different files. Just to make this clear,

Line **000001** is line 1 of file **AAA.TXT**
Line **000002** is line 2 of file **AAA.TXT**
Line **000003** is line 3 of file **AAA.TXT**

Line **000004** is line 1 of file **BBB.TXT**
Line **000005** is line 2 of file **BBB.TXT**
Line **000006** is line 3 of file **BBB.TXT**

Line **000007** is line 1 of file **JJJ.TXT**
Line **000008** is line 2 of file **JJJ.TXT**
Line **000009** is line 3 of file **T.JJJ.TXT**

The reason the line numbers are consecutive is that, internally, SPFLite treats the *entire* edit session as though it were a *single file*. That is because it has read-in every file into memory in a single location. By doing this, it makes it possible to issue ordinary SPFLite editing commands that end up affecting multiple files.

If you are following the example and look closely, you will see that the file names in each **=FILE>** line are in the same order as they were in the File List they were selected from using the **M** line command. This means that if you want or need the files in your multi-edit session to be in some particular order, you must sort the directory list or File List as you need it to be, prior to starting the multi-edit session.

The color of the multi-edit tab label is handled the same as regular file tab labels are, and if anything is updated, the color will change, if you have set up your color scheme to do this. Let's say that unmodified files will have a tab that looks like **(M-Edit)** in blue. You will also notice on the Status line, where it would normally say **Edit**, instead it shows **3 Edit**. This means you have 3 files involved in this edit session.

Notice in the display above, on line 3, the word **THREE** is misspelled as **THRE**. Let's correct it, by typing over it and pressing Enter:

(Multi-Edit) - SPFLite(v8.0.4129)

File Manager (M-Edit)

Command > _

Scroll > HALF

```
***** **** Top of Data ****
=FILE* D:\Documents\Test Data\AAA.TXT
000001 LINE ONE OF FILE AAA.TXT
000002 LINE TWO OF FILE AAA.TXT
000003 LINE THREE OF FILE AAA.TXT
=FILE> D:\Documents\Test Data\BBB.TXT
000004 LINE 1 OF ANOTHER FILE - BBB.TXT
000005 LINE 2 OF ANOTHER FILE - BBB.TXT
000006 LINE 3 OF ANOTHER FILE - BBB.TXT
=FILE> D:\Documents\Test Data\JJJ.TXT
000007 LINE 1 OF YET ANOTHER FILE - JJJ.TXT
000008 LINE 2 OF YET ANOTHER FILE - JJJ.TXT
000009 LINE 3 OF YET ANOTHER FILE - JJJ.TXT
***** **** Bottom of Data ****
```

3 Edit 1* | 2014-05-10 13:59:14 | Lines: 9 | Cols 1 to 80 | Bnds: MAX | OVR | T | CS | S+ |

What changes?

- The first **=FILE>** marker changes to **=FILE*** to show that **this file** has been modified.
- The file tab will change color (depending on your color setup); let us say it now shows **(M-Edit)** in red; that means that **this multi-edit session** has changed.
- The Status line will change from **3 Edit** to **3 Edit 1***. The notation **3 Edit 1*** means, you have **3 files** involved in the multi-edit session, and **1** of them has been **modified**.

Note: When the sequence area width is set to 5, the **=FILE>** and **=FILE*** markers are displayed as **FILE>** and **FILE***, respectively.

Suppose you wanted to change the word **FILE** to **DATASET** in all these files. How would you do it? Here is where the power of multi-edit comes into play. Within this edit session, you see 9 lines of data. To SPFLite, these 9 lines are treated internally as if they were a *single file*, rather than three. Because the **=FILE>** markers are **not data**, the editing commands you normally use in SPFLite (with very few exceptions) simply ignore these marker lines. This means that you can change the data for **every file** (or, as many of them as you'd like to) with a *single command*. Here's how you change the word:

Command > CHANGE FILE DATASET ALL

It's just that simple!

Here is what your edit session will look like after you do this:

```

***** **** Top of Data ****
=FILE* D:\Documents\Test Data\AAA.TXT
==CHG> LINE ONE OF DATASET AAA.TXT
==CHG> LINE TWO OF DATASET AAA.TXT
==CHG> LINE THREE OF DATASET AAA.TXT
=FILE* D:\Documents\Test Data\BBB.TXT
==CHG> LINE 1 OF ANOTHER DATASET - BBB.TXT
==CHG> LINE 2 OF ANOTHER DATASET - BBB.TXT
==CHG> LINE 3 OF ANOTHER DATASET - BBB.TXT
=FILE* D:\Documents\Test Data\JJJ.TXT
==CHG> LINE 1 OF YET ANOTHER DATASET - JJJ.TXT
==CHG> LINE 2 OF YET ANOTHER DATASET - JJJ.TXT
==CHG> LINE 3 OF YET ANOTHER DATASET - JJJ.TXT
***** **** Bottom of Data ****

```

3 Edit 3* L 000001 C 19 Lines: 9 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0027

Again, what changes?

- Now, *all* ==CHG> markers have changed to =FILE* because every file has been modified
- The file tab continues to show (M-Edit) in red to reflect that at least one file was modified.
- The Status line will change from 3 Edit 1* to 3 Edit 3*. The notation 3 Edit 3* means, you have 3 files involved in the edit session, and all 3 of them have been modified.

You can get rid of the ==CHG> markers by issuing a **RESET** command, just as you would in a regular file.

Storing data during **SAVE** and **END** processing

Now that we have changed the data in all these files, let's save them. What do you need to do to save every one of these files? Just say **SAVE** like you do for a regular file. You can type in the **SAVE** command, or issue it from a mapped key. When you do this, SPFLite is aware that it's inside a multi-edit session, and it saves every file to the file location you see on each =FILE> line.

When you say **SAVE** in our example, you will see the message: **3 files saved**. All of the *FILE> markers revert back to =FILE> to show each file no longer has unsaved changes.

Will **SAVE** always save *all* files? Yes, just as a **SAVE** command on a single-file Edit will save the file regardless of its modified status, so does **SAVE** in a multi-edit session.

When you **END** a multi-edit session and there are unsaved changes in any file, these will get saved according to the AUTOSAVE options in effect individually by file. **END** will only save *modified* files.

Note: When you **END** a multi-edit session, the file type, and thus the PROFILE, for *each individual file*, is **individually respected**. That means some files might have different AUTOSAVE options. It's up to you how you want to handle this in cases where you do a multi-edit of files having different PROFILE settings.

You can also **CANCEL** a multi-edit session, discarding any unsaved changes that may exist across all the files you have open, just as you could for a regular edit session.

Simulating a multi-browse with **SAVEAS**

There is no Browse equivalent of multi-edit; that is, there is no "multi-browse" command. However, there may be times when you might wish to temporarily work with a set of files (the way multi-edit does) without risking changes to any of the files (the way a Browse session does).

To accomplish this, you can begin a multi-edit session as normal, and then issue a **SAVEAS** command with no operands. This will bring up a directory-browse dialog, where you choose a directory (or create one) where all the files in your multi-edit session are saved. Once you do this, you will begin a new multi-edit session in which each **=FILE>** line will have the same file name and the (new) directory you have chosen.

You can do the **SAVEAS** selecting an existing, non-empty directory, but if any files within that directory have the same basic file names as the ones you are trying to save, the **SAVEAS** operation will not be performed and no files will be saved. (**SAVEAS** will not write over existing files.) Thus, the normal and preferred procedure is to create a new empty directory (or take steps to ensure that the **SAVEAS** target directory is empty ahead of time) as part of your **SAVEAS** process.

When you begin a multi-edit session from a File List using files originating from more than one directory, SPFLite permits you to have files in different directories that have the same basic file name in the same multi-edit session. However, if you intend to do a **SAVEAS** command, all of the basic file names involved must be unique, since the **SAVEAS** command will save all of the files from your multi-edit session into the same directory. If they are not unique, you can continue to use your multi-edit session, but you will not be able to issue a **SAVEAS** command.

When you do a **SAVEAS** command, every file currently in the multi-edit session is saved to the new location, even files not currently in a modified state.

Adding and removing files from a multi-edit session

If you issue a **D** line command on a **=FILE>** marker, the marker line and all lines associated with that file will disappear. **This does NOT delete the file itself**, but it removes the file from the multi-edit session. Any unsaved changes that may exist for that file will be discarded, the same as when a **CANCEL** command is issued. Because a **D** line command on a **=FILE>** marker removes the file from the multi-edit session but does not delete it, you can think of using a **D** line command this way as if it meant “**detach**” instead of “**delete**”.

(When you use a **D** line command on ordinary data lines that *follow* the **=FILE>** line, the **D** will delete the data, just as it does in a regular file.)

If you wish to bring in another file to an existing multi-edit session, issue an **MEDIT** primary command. This will create a new **=FILE>** separator line after the last data line in the last file, and then the data lines from the newly added file will appear after that. If you started editing a single file as an ordinary edit session, **MEDIT** will convert the edit session to a multi-edit session and then add the file to the end.

Line commands permitted on **=FILE> lines**

As mentioned, you can use the **D** line to remove a file from the multi-edit session. The following line commands are also available on the **=FILE>** separator line:

- **X** will exclude all the lines belonging to the file. It will not exclude the **=FILE>** separator line itself. You will see the excluded-line placeholder in place of the data lines for that file, but the **=FILE>** separator line will always be visible and cannot be excluded.
- **S** will unexclude all the lines belonging to the file, undoing what an **X** line command on a **=FILE>** line did.
- **A** and **B** can be put on this line if you are moving data lines after or before this point.

Note that a **=FILE>** line itself cannot be moved or copied, either directly or indirectly, or typed on, nor can it be changed manually or through primary or line commands.

(Users of mainframe ISPF can compare how this works to the dashed line that separated an ISPF 3270 split-screen session. You could reposition the dashed line, but you could not get rid of it, unless you closed out the split-screen session. The principle here is very similar. In fact, since SPFLite does not support “split screen mode” as mainframe ISPF does, but does support Multi-Edit, you could use Multi-Edit to partially simulate an ISPF-like split-screen session, if you were so inclined.)

The **(ClipName)** and **(ClipPath)** keyboard primitive functions in Multi-Edit

These functions normally return the file name or full path name of the edit file. In a multi-edit session, there are multiple files and thus multiple names. In this case, the **(ClipName)** and **(ClipPath)** keyboard primitive functions will place into the clipboard the name of the particular file where the cursor is located at the moment, whether it is on a **=FILE>** line or is on any data line associated with that **=FILE>** line. If the cursor is not in a “file area” when you use these functions, such as on the primary command line, it is an error, and the clipboard will not get anything stored into it.

The **(ClipName)** and **(ClipPath)** keyboard primitive functions have also been enhanced to return the file name of a file that the cursor is on if you are in the File Manager.

Creating and using orphaned lines

Normally, the Top of Data line in a multi-edit session is immediately followed by a **=FILE>** separator line and then line 1 of the first file. If you copy one or more data lines after the Top of Data line, these lines will have no **=FILE>** separator preceding them. These lines do not belong to any file at all, but are “**orphaned**” lines. During a **SAVE** or **END** command, orphaned lines are not saved.

You can use orphaned lines as temporary data, perhaps data that you wish to edit and then copy or move to other parts of the multi-edit session. Having orphaned lines is not an error; you just need to be aware of how SPFLite treats these lines.

Navigating from one file to another

The **LOCATE** command has been extended with a *locate line-type* of **FILE**. You can **LOCATE** to the **FIRST**, **LAST**, **PREV** or **NEXT =FILE>** line using this command, by specifying **LOCATE FILE FIRST**, etc.

As you do, you will see a message such as, **File 1 of 3 found**. You can use this information to help keep track of where you are in the multi-edit session, which may be useful if you have many files involved.

If you need to find a certain file number and you have many files (let's say, 100 of them) there is no command to directly find the sixty-third file, for example. However, let's say you have the F5 key mapped to **RLOCFIND**, which will repeat the last **LOCATE** or **FIND**, whichever has happened most recently, and the auto-repeat checkbox for that key is checked. You can first issue a **LOCATE FILE FIRST**, and you will see the message (in this example), **File 1 of 100 found**. To find file 63, just hold down the F5 key until you (eventually) see the **File 63 of 100 found** message.

(In case you pass it up, you could also have Shift-F5 mapped to **RLOCFIND REVERSE** (for example). Then, just hold down the Shift key and press F5 to go backwards until the right file number shows up in the message, and there you are. If you don't have a key for **RLOCFIND REVERSE** set up, just slow down when you get near the right file number so you don't pass it up.)

The command **LOCATE FILE** must not be combined with **ALL** or **MX**.

Dealing with external changes while an edit is in progress

SPFLite will keep track of every file involved in a multi-edit session, so that you are protected from, or notified about, external changes to any file in your multi-edit session, the same as you would for an individual edit file in the same situation.

Treatment of unqualified file names on primary commands

When an unqualified file name appears in a primary command issued from a multi-edit session, SPFLite must determine the directory to look at when resolving the file name.

The method it uses is the take the file path of the **last** loaded file as the assumed directory. This means that

commands like **MEDIT**, **COPY**, etc. that take file names will look in the directory where the last-loaded file was found to find files that are not fully-qualified on the command line.

This should be taken into account when you have multi-edit sessions that involve files from more than one directory, which can only happen if you started your multi-edit session from a File List display instead of from a directory list display.

Read Only files and MEDIT

SPFLite does not currently support Read Only files participating in Multi-Edit sessions. If you want to use a Read Only file within **MEDIT**, you must remove the Read Only attribute, or copy the file to another name, and ensure that the copy of the file is not Read Only.

What else can you do in a multi-edit session ?

Basically, anything that you could do in a regular edit session, multiplied by the power of acting on multiple files at once.

The possibilities are intriguing:

- Exclude, Flip, and Show to manage the exclusion status of the lines
- **SORT** across multiple files at a time (this **will** work, but the results may surprise you; be certain this is what you intended)
- Use tags to identify and edit lines across multiple files
- Use Power Typing to interactively update data lines in parallel, across multiple files
- **SUBMIT** jobs consisting of multiple files
- Print multiple files at one time with a single **PRINT** command

Maybe you can come with interesting techniques of your own. If you do, please let us know!

Working with Line Labels

Contents of Article

[Introduction](#)

[Relative line label notation](#)

[Relative label ranges with AND and OR](#)

[Temporary line labels, also known as line-number pseudo labels](#)

[Using the single character line range arguments](#)

[Using Labels and Tags with the LINE Command](#)

[A handy key mapping using a label for a placeholder](#)

Introduction

Users of SPFLite and/or ISPF are familiar with line labels. A *line label* is a `.` dot followed by one or more letters, and appears in the sequence area.

Note: The line number sequence area size on the edit screen is adjustable from 5 to 8 positions. This means the maximum size of a label or tag that you can type into a sequence area is from 4 to 7 letters, after the leading dot or colon. Internally, SPFLite maintains an 8-position sequence area, and only displays as much as will fit.

This means that labels and tags on the primary command line can actually use the maximum internal size (a dot or colon plus up to 7 letters). When you do this, only part of the label or tag is displayed, but the entire value is present. That should not pose a problem if the labels are being managed by macros, but if you enter such labels manually, be careful you don't confuse yourself, since you won't see the entire label displayed on the edit screen.

Labels are associated with the data line itself, not the particular line number they are on at the moment. So, if lines are inserted or deleted prior to the labeled line, or if the labeled line itself is moved, the label stays with the line. If there are two lines, one with label `.AA` and one with label `.BB`, ISPF allows you to issue primary commands like this:

CHANGE ABC DEF ALL .AA.BB

and all lines from `.AA` to line `.BB`, inclusive, will be changed. It doesn't matter if `.AA` comes before `.BB` or vice versa; the **CHANGE** will work the same way.

If you only want to change a single labeled line by itself in ISPF, you have to repeat the label, like this:

CHANGE ABC DEF ALL .AA.AA

but SPFLite allows the second label to be omitted if it's the same as the first. So the same thing can be done with just:

CHANGE ABC DEF ALL .AA

There are several "special" line labels that are available:

.ZFIRST (or **.ZF**) to indicate the first data line of the file

.ZLAST (or **.ZL**) to indicate the last data line of the file

.ZCSR to indicate the line in the data area where the cursor is located

.ZFIN to indicate the last (most recent) line found by a FIND command

.ZLOC

to indicate the last (most recent) line found by a LOCATE command

For .ZCSR, the cursor must be on a data line (or the sequence area of a data line) in the edit screen.

For .ZFIND and .ZLOC, you must have issued at least one prior FIND, CHANGE or LOCATE command for the current file.

Otherwise, it is illegal to use these three special line labels, and you will get an "undefined label" error message if you try.

The handling of the .ZCSR special label is mostly compatible with ISPF. If you use .ZCSR in ISPF when the cursor is not in the data area, the line is simply not found (the ISPF FIND and CHANGE commands will simply report Bottom of Data) rather than reporting an error. This minor difference should not pose any difficulties for you.

Relative line label notation

A *relative label* defines a region of lines in the edit file with respect to a label. You could say that a simple label itself defines a *basic* region consisting of the very line that the label is on. There are now additional regions available, using the following syntax:

.ABC	the line .ABC
. \ABC or .<>ABC or .¬ABC	all lines other than line .ABC
. >ABC	all lines after line .ABC
. >=ABC	the line .ABC and all lines after it
. <ABC	all lines before line .ABC
. <=ABC	the line .ABC and all lines before it

Because of these definitions, it turns out that, individually, .>=ZF and .<=ZL both mean the same as .ZF .ZL, which is the entire file.

You may have noticed that when you issue a command like CHANGE ABC DEF ALL .AA .BB, the line range you are implying is the same as .>=AA .<=BB if you were to use the notation above. When you use **two** relative labels in a command, there are some order-dependency considerations to keep in mind for them to work correctly:

- When two labels are present and a label of the form .>ABC or .>=ABC is present, it must be the first label.
- When two labels are present and a label of the form .<ABC or .<=ABC is present, it must be the second label.
- When a label of the form .\ABC or .<>ABC or .¬ABC is present, it must stand alone; there cannot be a second label on the command.

When two labels .ABC and .DEF are defined, and you want to use them on a command such that one is a regular label and one is a relative label, one of the following forms must be used (in this order):

.ABC .<DEF	same as .>=ABC .<DEF
.ABC .<=DEF	same as .>=ABC .<=DEF
.>ABC .DEF	same as .>ABC .<=DEF

.>=ABC .DEF

same as .>=ABC .<=DEF

Relative label ranges with AND and OR

There may be times when you don't wish to be limited by the order dependency issue discussed above. Assume that .AA precedes .BB in the file. Then, the range .>AA .<BB actually defines an *intersection* of lines, in which all lines that are both *after* line .AA **and** *before* line .BB are intended. You can explicitly show this in SPFLite with the command:

```
CHANGE ABC DEF ALL .>AA AND .<BB
```

The AND keyword is actually allowed in SPFLite, but it's always assumed if you don't say it.

Now, if you had a 300-line file, with .AA on line 100 and .BB on line 200, then .>AA .<BB represents the "interior" of the file from lines 101 to 199. Suppose you wanted to reference the "outer edges" of the file; that is, all lines *other* than these lines? It would be as if you were to say,

```
CHANGE ABC DEF ALL NOT (.>AA .AND .<BB) -- illegal syntax
```

However, SPFLite doesn't *support* a **NOT** operator, or line ranges in expressions. Suppose you tried to use the ranges without any options, like this:

```
CHANGE ABC DEF ALL .<=AA .>=BB
```

Since the **AND** option is assumed, and since .AA is before .BB in the file, there isn't a single line in the file that would meet *both* of these requirements *at the same time*, and so no lines would be changed.

If .AA is on line 10, and .BB is on line 20, then the line range .<=AA .>=BB is asking for all lines where the same line number is **both** ≤ 10 **and** ≥ 20 , at the same time. That would be a neat trick if you could do it. SPFLite isn't that tricky, and so it will report **Line range illogical, represents 0 lines**.

What SPFLite *does* have is an **OR** facility, which defines a *union* of lines that meet either one of two conditions. Let's rewrite the command above:

```
CHANGE ABC DEF ALL .<=AA OR .>=BB
```

Like all keywords in SPFLite, the **AND** and **OR** are case-insensitive.

Now, we have a line range consisting of two independent ranges. This command will change any lines found **either** in the range of 1 to 100, **or** in the range of 200 to 300, in our example.

Note that when **AND** is either *used* or *implied*, the order of your labels doesn't matter (it has to be that way to be ISPF compatible). That means the regular labels you are used to, like .A.B mean the same as .B .A just like always.

This lack of order dependency only applies when both labels are **regular** labels like .A and .B. When one or both of these labels are **relative** labels like .<A, .<=B, .>C or .>=D, the ordering rules described above apply.

When **OR** is used, the line number associated with the first label must be **before** the line associated with the second label. This is a safeguard to make sure the line range you specify makes sense.

Temporary line labels, also known as line-number pseudo labels

Sometimes you may need to use a command like **CHANGE** on a known range of line numbers, but the **CHANGE** command doesn't accept actual line numbers as operands to indicate line ranges. There *are* other ways to do it, but often what happens is that you end up doing this:

- Temporarily place one-time labels like .A and .B on the ends of the desired range
- Issue the **CHANGE** or other command, referencing .A and .B
- Go back and get rid of the temporary labels .A and .B

Suppose you knew ahead of time that the .A label will go on line 100, and the .B label will go on line 200. With SPFLite you no longer have to “invent” these .A and .B labels. You can use *temporary line labels*, also known as *line-number pseudo labels*. Such a pseudo label is a . dot followed by a line number. The line number is handled numerically, not as a string. So, for example, a pseudo label of .100 or .00100 both refer to the same line.

You can use a pseudo label in commands just like a real one, including relative labels such as .\100 and .<100.

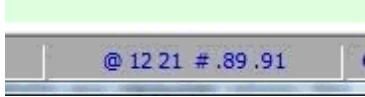
The only restriction is that if you attempt to **RESET LABEL** using a pseudo label, you can't really reset the label *itself*, since there is no actual label called .100. What happens is that if you issue the command **RESET LABEL .100** it will remove any *ordinary* label that might happen to exist on line 100.

A pseudo-label like .100 is not a real label, and never exists in the sequence area of a data line, so there is no need to issue a **RESET** command to get rid of it. There is no label needing to be blanked out.

Using the single character line range arguments.

When text is selected using mouse drag or via the keyboard Shift-Arrow keys, a block of text is defined and highlighted on the screen. The line and column range of this selected area is remembered and can be used to quickly re-select the area for further processing if desired.

When an area is selected, the line and column references of the area are displayed in a Status Bar box for reference. That portion of the Status Bar would appear as



where the @ and the two numbers following represent the **column** range of the selected area, and the # and the two operands following represent the **line** range of the selected text.

The significance of the @ and # characters is both to identify which pair of numbers is which, but also to remind you that these single character values can now be used on the Primary command line for commands that accept line/column range operands.

For example, using the above displayed sample, a command line of **PRINT #** would be treated as if it were entered as **PRINT .89 .91** to print the selected line range.

A change command **CHANGE AAA BBB @ #** would be treated as **CHANGE AAA BBB 12 21 .89 .91**. i.e. Change AAA to BBB on lines 89 to 91 between columns 12 and 21.

The values displayed are persistent and will remain available for repeated use on commands until a new, replacement text selection is made or a **RESET** command is issued..

Suspending the Range

You may temporarily suspend the @ and # processing on the command line by simply left-clicking on the Status Bar box displaying these values. When you do the box will switch to



This display indicates that an area has been selected and is currently inactive. At this point use of @ and # on

the command line will **not** trigger substitution of line/column values.

A left-click on this box at this point will switch back to the normal select display, and will also re-select the text area on the screen for confirmation.

Using Labels and Tags with the LINE Command

You can manage line labels and line tags using the LINE primary command. For example, to put a label of .ABC on line 10, you can issue the command:

```
LINE '.ABC' .10
```

Remember to quote the label, because in this example, the first operand `'.ABC'` is **what** goes into the sequence area, and the second operand `.10` is **where** the first operand is placed.

When doing a normal edit, there is usually no need to do such a thing; you would just put the label on the line directly. However, when running a programmable macro the LINE primary command is the only way to do this. You would need to 'wrap' the LINE primary command in an **SPF_CMD** function call in your **.MACRO** file, like this:

```
SPF_CMD ("LINE '.ABC' .10")
```

It is possible to set, clear and toggle both line labels and line tags from the LINE primary command. The possible options are as follows:

<code>LINE '.label'</code>	<i>-- Enter label into sequence area</i>
<code>LINE '..label'</code>	<i>-- Toggle label; enter label if no label present, -- else clear any label that is present</i>
<code>LINE '.'</code>	<i>-- Clear any label that may be present</i>
<code>LINE '..'</code>	<i>-- Clear any label that may be present; same as <code>LINE '..'</code></i>
<code>LINE ':tag'</code>	<i>-- Enter tag into sequence area</i>
<code>LINE '::tag'</code>	<i>-- Toggle tag; enter tag if no tag present, -- else clear any tag that is present</i>
<code>LINE ':'</code>	<i>-- Clear any tag that may be present</i>
<code>LINE '::'</code>	<i>-- Clear any tag that may be present; same as <code>LINE ':'</code></i>

See [LINE - Apply Line Command](#) and [Working with the LINE Primary Command](#) for more information on the LINE command.

See [Special Line Commands](#) in [Using the KEYMAP Dialog](#) for more information on performing these same actions within a KEYMAP definition.

A handy key mapping using a label for a placeholder

Here is a simple technique using line labels you might find useful. Often you may need to remember where you were in some particular location in a file, then do something which changes the location or appearance of the screen, such as a **FIND**, **LOCATE** or perhaps a **RESET** after an **EXCLUDE** command has hidden data in several locations. You would then like to get back to where you were.

The technique works like this. You map the **Alt 3** key to "make a note" of where you had been in the file, and you

map **Alt 2** to go back to where you were at previously. Since the '3' key has the # pound sign (also called the 'note sign') and the '2' key has the @ At sign, you can use these two keys to help you remember what each of these keys are for. (It is kind of a **pun** - but it's also an effective way to remember them.)

The mappings are as follows. The technique uses Alt 3 to set (or clear) a label of **.NOTE**, and Alt locates that **.NOTE** and aligns the screen so the label is at the top of the display.

Alt 2 = **LOCATE .NOTE TOP**

Alt 3 = **(CondLineNo) { ..NOTE }**

For Alt 3, the **(CondLineNo)** function allows the setting of the **.NOTE** label even when the cursor is on the primary command line; if so, the label is set on the first data line of the display. The **{ ..NOTE }** will store a label of **.NOTE** if no label is present on the cursor line, or will clear it if one is already there.

Working with Line Tags

Contents of Article

- [Introduction](#)
- [Example: Changing tagged lines](#)
- [Why use tagged lines ?](#)
- [Relative line tag notation](#)
- [Line label and line tag co-location](#)
- [Primary-command tag usage](#)
- [Primary-command tag examples](#)
- [The TAG primary command](#)
- [Using tags for line control ranges](#)
- [The DROP and KEEP commands](#)
- [Using Labels and Tags with the LINE Command](#)

Introduction

SPFLite contains an exciting new technology extension to ISPF: **Line Tags**. You will note a number of similarities between a line tag and a line label.

Note: **User Lines** provide an alternate means to segregate lines into two groups, known as **U lines** and **V lines**, where U lines are a set of lines of particular interest to a user at a given point in time, and "non-User" lines are everything else. The terms **User Line** and **U line** mean the same thing; and, **non-User Line**, **V line** and **ordinary data line**, all mean the same thing.

User lines have a number of similarities with excluded lines, but User lines are not hidden and then revealed the way excluded lines are. If you only need two types of lines to work with, using User Lines may be simpler than using tagged lines.

See [Working with User lines](#) for more information.

A *line tag* is a : colon followed by one or more letters, and appears in the sequence area.

Note: The line number sequence area size on the edit screen is adjustable from 5 to 8 positions. This means the maximum size of a label or tag that you can type into a sequence area is from 4 to 7 letters, after the leading dot or colon. Internally, SPFLite maintains an 8-position sequence area, and only displays as much as will fit.

This means that labels and tags on the primary command line can actually use the maximum internal size (a dot or colon plus up to 7 letters). When you do this, only part of the label or tag is displayed, but the entire value is present. That should not pose a problem if the tags are being managed by macros, but if you enter such tags manually, be careful you don't confuse yourself, since you won't see the entire tag displayed on the edit screen.

Tags are associated with the data line itself, not the particular line number they are on at the moment. So, if lines are inserted or deleted prior to a tagged line, or if the tagged line itself is moved, the tag stays with the line.

You can enter a tag on a line by typing it in the sequence area, and you can remove by blanking it out, just like you would with a label. However, as you will see, there are other, more powerful ways of tagging and untagging lines.

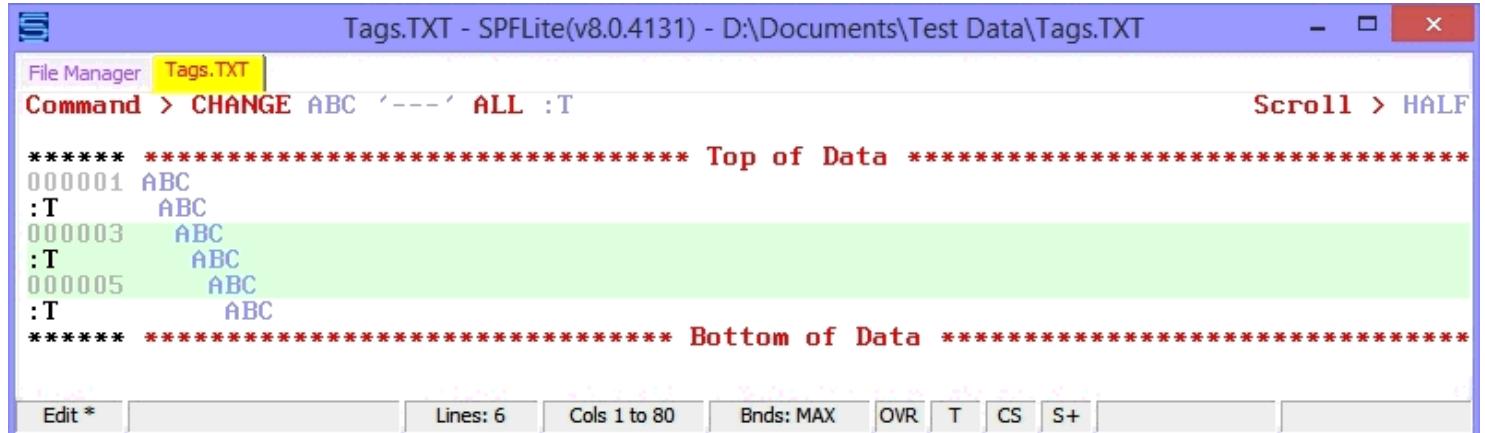
The major difference between a tag and a label is that the **same** tag name can appear on **multiple** lines. Like labels, tags can be used on primary commands to reference a set of lines to be acted upon.

If you like, you can think of the single dot that precedes a label, and the two dots that make up the colon preceding a tag, to help you remember that a label applies to a single line, while tags apply to multiple lines (usually).

A given tag name usually will end up appearing on multiple lines, because that's where tags derive their power. But there is no reason why a particular tag name can't be on just a single line.

Example: Changing tagged lines

Let's say we have the following file, and manually entered a tag name of `:T` on the lines shown, and then issue a **CHANGE** command that uses those lines:

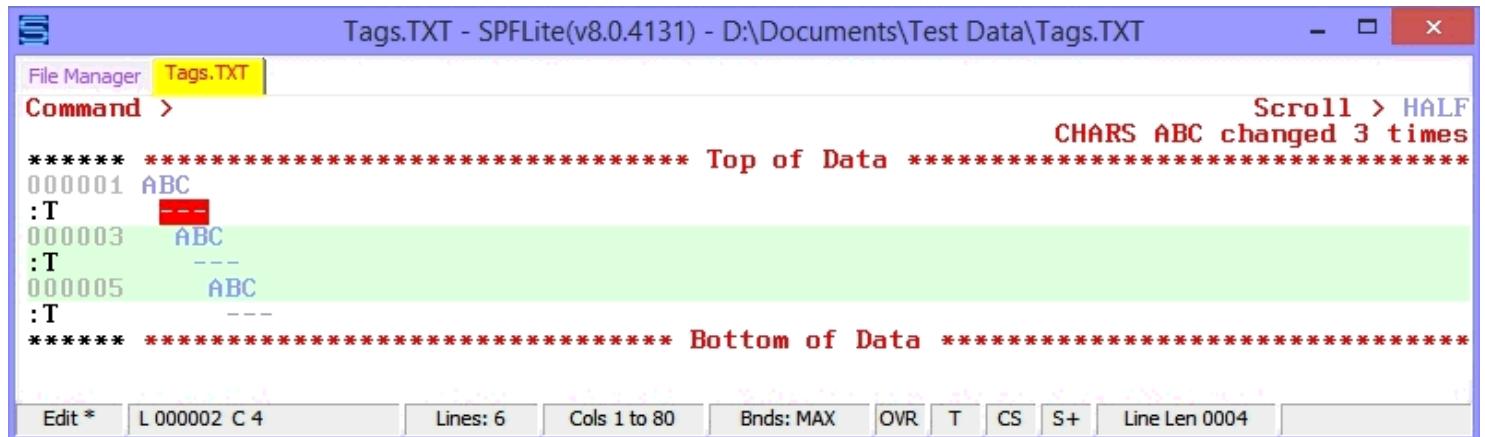


```
Tags.TXT - SPFLite(v8.0.4131) - D:\Documents\Test Data\Tags.TXT
File Manager Tags.TXT
Command > CHANGE ABC '---' ALL :T
Scroll > HALF

***** **** Top of Data ****
000001 ABC
:T ABC
000003 ABC
:T ABC
000005 ABC
:T ABC
***** Bottom of Data *****

Edit * Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0004
```

When you are done, the file display will look like this:



```
Tags.TXT - SPFLite(v8.0.4131) - D:\Documents\Test Data\Tags.TXT
File Manager Tags.TXT
Command >
Scroll > HALF
CHARS ABC changed 3 times

***** **** Top of Data ****
000001 ABC
:T ---
000003 ABC
:T ---
000005 ABC
:T ---
***** Bottom of Data *****

Edit * L 000002 C 4 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0004
```

Tags are *persistent*. If you are editing this file and have a PROFILE option of **STATE ON**, and these tags are present, you can close this file and then reopen it later, and again, the three `:T` tag names will still be there.

Why use tagged lines ?

Well, consider why *excluded* lines are used. (Review "[Working with excluded lines](#)" section as needed for more information). Excluded lines are a way to sub-divide, or *partition*, a file into two groups or subsets: the *excluded* group, and the *unexcluded* group. You can use **X** or **NX** on commands to select between these two groups.

But suppose you needed *three* or *four* groups – or maybe, *lots* of groups. What if you needed to do extensive work on one or more of these groups, including things like **FIND** and **CHANGE** that normally unexclude lines when string are found and changed, and thus might change a line from being in one group to being in another? Excluded lines don't help much any more in cases like this. But *tagged* lines **do**. Tagged lines are a much more powerful facility for managing your data. You will see this as the discussion proceeds.

You may wish to review [Case Study: Implementing Bookmarks](#) in [Keyboard Macros](#) for an example that combines line tags and key mapping.

Two points to note here:

- First, because tags are more powerful than exclusions, there is more involved in using them. There are additional commands and considerations. These are discussed in detail. Be prepared to do a little studying up before you grasp the whole picture.
- Second, labeled lines, tagged lines, excluded lines **and** User Lines can be **combined** for even more powerful data selection capabilities. Users involved in activities like data management will see possibilities to perform certain types of *data mining* operations on text data, using labeled, tagged, excluded, and User lines, which could be quite useful in inspecting or analyzing very large text files.

Relative line tag notation

Like labels, tags may take a *relative* notation, although it's a bit simpler. (The relative label formats `< <= >` and `>=` are not used on tags.)

A *relative tag* defines a collection of lines in the edit file with respect to a tag. The following relative line tags exist:

<code>:ABC</code>	all lines tagged with <code>:ABC</code>
<code>: \ABC</code> or <code>:<>ABC</code> or <code>:¬ABC</code>	all lines <i>other than</i> lines tagged with <code>:ABC</code>
<code>: ZALL</code>	all lines having any tag whatsoever regardless of tag name
<code>: \ZALL</code> or <code>:<>ZALL</code> or <code>:¬ZALL</code>	all lines having no tag present

Note that `:ABC` is just an *example* here, whereas `:ZALL` is literally specified as a special tag name, reserved to collectively reference *all* lines having a line tag of any kind. It is illegal to actually enter a tag of `:ZALL` into the sequence area of a line.

The **Z** prefix on **ALL** comes from conventions IBM uses for ISPF reserved names, such as `.ZFIRST`.

Line label and line tag co-location

The *label* of a line, and the *tag* of a line, are two separate, independent attributes of a line (along with the exclusion status of a line). This means a line may be labeled or not labeled, tagged or not tagged, and excluded or not excluded, all independently of each other.

When a line has both a label and a tag at the same time, the screen display will only show the *label*, since a label is always unique, but a tag is normally not unique. What happens is that the usually-blank column between the sequence area and column 1 of the data line will contain a `:` colon as a reminder that there is a tag defined for that line. You can't see the tag, but at least you know one is *there*.

Sometimes that usually-blank column is called the *gap column*. As with character positions in the sequence area, the gap column has no column number.

Let's say we have the following file, and we manually entered a tag name of `:T` on lines 2, 4 and 6, and then on line 4, then we also entered a label of `.AA`. The file display will look like this:

File Manager Tags.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 ABC
:T ABC
000003 ABC
.AA : ABC
000005 ABC
:T ABC
***** **** Bottom of Data ****
```

Edit * L 000004 C 4 :T Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006 @ 44 #.4.4

Notice the : colon after label .AA on line 4. Suppose you put the cursor on line 4, column 4, where the A of ABC is underlined above. The status line for the line/column indicator will look like this. Notice that in addition to the line and column, you will see the hidden tag name:

L 000004 C 4 :T

The status line *only* shows tag names when the tag for a line is *hidden*. When there is no tag/label co-location going on, the line/column display will not display tag names, even for lines that *have* tags.

Why not just show tag names all the time in the line/column display, instead of only when there is co-location? Mainly to keep the status line as simple and uncluttered as possible. We don't want to distract you with information you can see anyway just by looking at the main screen.

Now, just to explore this idea, suppose we were to set up the following keyboard mapping:

Alt-period = { . }
Alt-semicolon = { : }

These are not defaults; you would have to do this key mapping yourself.

This will allow us to use the keyboard to clear the sequence area of labels or tags – even in cases where you couldn't directly get to a tag, or type over it, because co-location was hiding it.

If you placed the cursor on line 4, and were to press **Alt-period**, the *label* and colon would go away, the tag would reappear, and line 4 would look like this:

File Manager Tags.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 ABC
:T ABC
000003 ABC
:T ABC
000005 ABC
:T ABC
***** **** Bottom of Data ****
```

Edit * L 000004 C 4 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006 @ 44 # .4.4

Whereas, if you placed the cursor on line 4, and were to press **Alt-seicolon**, the *tag* and colon would go away, and the line 4 would look like this:

File Manager Tags.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 ABC
:T ABC
000003 ABC
.AA ABC
000005 ABC
:T ABC
***** **** Bottom of Data ****
```

Edit * L 000004 C 4 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006 @ 44 # .4.4

In both cases, once the label/tag co-location is no longer present, the : colon in the gap column and the extra :T tag name that was present in the status line's line/column display will disappear.

Primary-command tag usage

In any primary command that references data lines, a tag name can be used instead of a label or pair of labels. Only one tag may appear (except for the **TAG** primary command, which has a special syntax.)

You can combine tags and labels on the same command to select lines. What happens is that the list of effective lines is the *intersection* of the labels and tags, so that only the tags that exist within the specified line label range are available to the command.

You can combine the **X** or **NX** option, and the **MX** or **DX** option, along with tags, on various commands, just as you could do if you were selecting lines or line ranges using labels.

If a command has both a tag name and **X** or **NX**, then lines with the given tag are selected limited by their **X** or **NX** status.

Primary-command tag examples

CHANGE ABC DEF ALL :T

Change text on all lines tagged with :T.

CHANGE ABC DEF ALL :T X

Change text on all lines tagged with :T, limited to excluded lines only. Changed lines will be unexcluded.

CHANGE ABC DEF ALL :T X DX

Change text on all lines tagged with :T, limited to excluded lines only. Changed lines will remain excluded.

CHANGE ABC DEF ALL :\T

Change text on all lines which are NOT tagged with :T.

CHANGE ABC DEF ALL :T .A.B X

Change text on all lines which are tagged with :T, limited to excluded lines only, that are within the range of labels .A and .B

CHANGE ABC DEF ALL :ZALL

Change text on all lines tagged with a tag of any kind.

CHANGE ABC DEF ALL :ZALL NX MX

Change text on all lines which have no tags of any kind and are not excluded. Changed lines will be made excluded afterwards.

The TAG primary command

The full potential of line tags is only realized when you use the **TAG** primary command to set, clear or toggle a given line tag *en masse* across many lines. It's a powerful command, and has many options. The [TAG](#) command description also discusses it, but let's go over it here.

Database users may see conceptual similarities between the **TAG** command and SQL's UPDATE / SELECT WHERE statements.

Here's the syntax of it:

```
TAG [ :tag-name ]  
[ search-string [ NF ] ]  
[ start-column [ end-column ] ]  
[ ON | OFF | TOGGLE | ASSERT | SET ]  
[ FIRST | LAST | NEXT | PREV | ALL ]  
[ PREFIX | SUFFIX | WORD | CHAR ]  
[ line-control-range ]  
[ X | NX ]  
[ MX | DX ]
```

That's a lot. We will take this one step at a time.

- The *tag-name* names the tag that is to be set, cleared or toggled. It's optional, because if you omit it, **TAG** can be used to *clear* all tags in a given line range when you are doing a **TAG OFF** or **TAG ASSERT** operation. Normally you will be using the *tag-name* operand., and for **ON** and **TOGGLE** it is required. The *tag-name* here represents the *target* of the **TAG** command's activity – what tag name will be used to set tags on with, for example. So, this tag is called the *target tag name*. The target tag, when used, **must** appear immediately after the **TAG** command name, and no where else.

Otherwise, it could possibly get confused with being part of the line-control range, and we wouldn't want *that* to happen, would we?

This is one of the few places where SPFLite has positional requirements for its command options. In most cases, you can put operands in any order you want, but not for this one.

- The *search-string* is used to locate particular lines for which you wish to set, clear, assert or toggle a line tag. If you omit the *search-string*, the command does not base its line-selection criteria on the *contents* of the lines, only their *location* in the file. A *search-string* is required for the **ASSERT** option. **Note:** A search-string on a **TAG** command is just like a search-string on a **FIND** command, which means the same kinds of SPF string types are permitted, such as **C**, **T**, **X**, **P** and **R** strings.
- If you use a *search-string*, you can use the **NF** option, which means *not found*. It works just like the **NFIND** command does, by selecting lines in which the *search-string* is **not** found. (Recall that the **NFIND** command can also be spelled as **NF**.)

Why didn't we make an **NTAG** command or something, to be "consistent" with **NFIND**? Since **TAG** is pretty complicated already, having *two* complicated tag commands just didn't seem like a good idea. Besides, most of what an **NTAG** command would have done is covered by the **ASSERT** option, so keep reading.

- The *start-column* and *end-column* are used to limit the columns where the *search-string* is searched for, and this works just like it does on a **FIND** command.
- The **ON**, **OFF**, **TOGGLE**, **ASSERT** and **SET** control what exactly the **TAG** command is going to do. Normally you are going to use **TAG** to *tag* things, and so **ON** is the default. You can abbreviate **TOGGLE** to **TOG**. **ON** and **OFF** are straight-forward. To **TOGGLE** a tag means (a) if a line has no tag, set the specified tag name, (b) if the line *has* the specified tag, clear that tag, and (c) if the line has a tag *other than* the specified tag, *leave that tag alone; don't change it and don't clear it*.

The **SET** option requires a tag name and a search-string. If the search string is found on the line (or, not found if the **NF** option is used), then the line is assigned the tag name. If search string is not found on the line (or, is found, if the **NF** option is used), then the line is cleared of any existing tags. **SET** may thus be used to assign a tag to a line range without have to pre-clear any existing tags in a separate step.

- The **ASSERT** option calls for some explanation. The short answer of what **ASSERT** does is this: It does exactly the same thing that **OFF** does, but the sense of the **NF** option is inverted. What in the world does *that* mean? Well, if you wanted to "assert" that a string existed on an *already* tagged line, what would it mean in practice? You would have to check to see if the string was there, and if it was, you'd leave the existing tag alone and not change it. But if the string wasn't there, you would erase the tag, since your "assertion test" had failed. So, a command like **TAG OFF 'string' NF** would accomplish this – tags are turned off if the string is not found. However, even though that is a perfectly good **TAG** command, it kind of looks like a "double negative" with "backward logic". It's just not the way people generally think about things. So, instead of this double-negative stuff, you can use "positive logic" and describe this operation in terms of what you *want* to be there, instead of what you *don't* want to be there. So, the command **TAG ASSERT 'string'** will do exactly the same thing as **TAG OFF 'string' NF**, without forcing your poor brain through contortions and back-flips trying to figure it out.

You can also use **NF** on **ASSERT**, though it would probably be used less frequently than **ASSERT** without **NF**. What would an **ASSERT NF** be used for? Say you had some tagged lines, and you wanted to make sure (assert) that some unwanted string was not there (**NF**). You *could* say **TAG OFF 'string'**, but if you have gotten into the habit of thinking in terms of assertions rather than just on/off, this format could be useful to you. Whether you use **OFF** or **ASSERT** is really a case of which makes more sense to you at the time.

- The **FIRST**, **LAST**, **PREV**, **NEXT**, and **ALL** options are like **FIND** and **CHANGE** as well. Normally you will want to act upon *all* lines that meet the search criteria of the command; that's where the power of the **TAG** command lies, after all. So, unlike other similar commands that default to **NEXT**, the **TAG** command defaults to **ALL**.

You should be aware that there is no "repeat tag" command that is comparable to **RFIND** or **RCHANGE**. If you choose to use **FIRST**, **LAST**, **PREV** or **NEXT** on **TAG**, it would ordinarily involve manual cursor placement if you wanted to repeat your initial single-line **TAG** command. However, you can use the **ITERATOR** Technique to get around this. See the [Application Note: The ITERATOR Technique](#) in [Command Macro Support](#) for more information.

- **PREFIX**, **SUFFIX**, **WORD** and **CHAR** are just like **FIND** and **CHANGE**, with **CHAR** being the default.
- The **X** and **NX** options are like **FIND** and **CHANGE**. They limit the selection of lines that **TAG** will operate on to either excluded or unexcluded lines.
- Tagging or untagging a line is comparable to finding a string with **FIND** or **CHANGE**; this is true whether you actually put a string value on the **TAG** command or not. So, tagging a line that is excluded will

unexclude it. If you want to modify that action, you can use the **MX** or **DX** options, which work the same way as they do on a **FIND** or **CHANGE**.

One important point to remember is that **TAG** will either set a tag on, off or toggle it, if the lines provided to **TAG** meet the criteria you specify. If they *don't* meet that criteria, the "tag status" of such lines is unchanged – existing tags will be left alone, and untagged lines will not get tagged, if they are not selected for processing by the **TAG** command.

So, for example, a **TAG :T ON 'string'** will tag lines having '*string*', but other lines that *already* had some kind of tag but didn't have '*string*' will not be "untagged" – they will be left alone.

Remember that **TAG ASSERT 'string'** is the same as **TAG OFF 'string' NF**, and **TAG ASSERT 'string' NF** is the same as **TAG OFF 'string'**. They both describe the same action, one with positive logic and one with negative logic.

Using tags for line control ranges

And now, saving the best for last ...

Notice the *line-control-range* operand. Just like other commands, this operand allows you to select one or more lines to operate on. If you don't specify a line-control-range, **TAG** will operate on a **CC** block, if you have defined one. If you haven't defined a **CC** block, **TAG** will operate on the entire file, subject to any other options you may have specified.

Otherwise, the *line-control-range* operand can be one or two line labels (including relative line notation), and/or it can be a line tag (or relative tag notation), and/or a **X** or **NX** excluded-line selection.

The line control range for **TAG** can have a *tag*? Yes.

The way this works is that a tag-name *here*, as a line-control-range, chooses which lines the **TAG** command will operate on. The tag-name at the *beginning* of the **TAG** command is then used to specify a *newtag* name to be given to all chosen lines. If any of those lines *already* have a tag, they will be replaced by the new tag, if the **ON** option is used or is defaulted.

If there can be *two* tag names in a **TAG** command, how do I know which is which? The tag immediately following the **TAG** command name is the value used for setting or toggling the new tag name on a data line. That tag is called the *target tag name*. A tag name appearing anywhere else is considered to be part of the line-control range, used for selecting eligible lines. Tag names in the line-control range can be also a relative tag like :**\ABC**, but the target tag cannot be; it has to be a simple tag name.

This is one of the few places in SPFLite where there are order dependencies on primary command operands. In most cases, the order you specify things doesn't matter, but here it does.

What could a **TAG** command with two tags present be used for?

Well, suppose you wanted to find all lines that had the set of strings GORT, KLATU, BARADA and NIKTO on them. **But**, the strings could be in *any order*, and might be separated by an unknown amount of text. You might be able to create a regular expression to find *two* such strings, but to find *four* of them, in any order, would require $4 * 3 * 2 = 24$ different possible orderings. Even if you're an expert at regular expressions, that would be pretty tough.

Here's where tags comes to the rescue. What we can do is first find all lines with GORT on them, and tag them as :G. Then, *only looking at the lines tagged with :G*, find lines with KLATU, and tag them with :K, etc. When you're done, you will have a set of lines tagged with :N which will contain the set of all four words, regardless of the order they appear on the line. You can call these strings **WORDS** on the **TAG** command, if that's what they are, just like on a **FIND** command. (We'll skip that here.) The sequence of TAG commands would work like this:

```
TAG :G 'GORT'  
TAG :K 'KLATU' :G  
TAG :B 'BARADA' :K  
TAG :N 'NIKTO' :B
```

If you wanted to, the initial TAG could be limited to a line range, like this:

```
TAG :G 'GORT' .ABC .DEF
```

or to a CC block, or to an excluded range, like this:

```
TAG :G 'GORT' X
```

but we'll continue with the discussion assuming that we're starting with the whole file, not just part of it.

The final tag :N can then be used to access only those lines having all four words. Now, suppose you wanted to view *only* these lines. You could say **X ALL :N** to exclude all lines that don't contain these four words, or you could say **DELETE ALL :N** to delete them.

Is it necessary to use all those tag names? Is there any way we can get by with using fewer names? Yes.

Don't try to just *alternate* tags, like using :A then :B then :A again, in order to achieve this. It won't work right.

Now, if you were to enter a **TAG** command that looks like **TAG :T 'string' :T**, it's asking the editor to select lines *already* tagged with :T that have 'string' on them, and then tag them with :T again. That won't do what we want here.

In plain English, it's called "going nowhere fast", and is just as useful as saying **CHANGE ABC ABC ALL**. However, SPFLite is nice enough to remind you of this fact, and will report, **No lines (re)tagged**.

However, you *can* selectively *remove* tags from lines in which a desired string is not found. So, you can reduce the number of tags needed to just one.

It's a little bit more complicated syntax, but you can use the **RETRIEVE** or **CRETRIEV** command (often mapped to F12) to pull up the more-complicated command and type over it; that will make it a little easier to do.

Let's revise the example above using just one tag:

```
TAG :A 'GORT'  
TAG OFF 'KLATU' NF :A  
TAG OFF 'BARADA' NF :A  
TAG OFF 'NIKTO' NF :A
```

That's a little better. The first **TAG** proceeds like the previous example. The remaining **TAG** commands successively "whittle away" at the list of lines represented by tag :A, so that any lines where the required words are not found (**NF**) no longer qualify as being in the :A list, and *their* :A tags get turned **OFF**.

If you format the command as shown above, you can almost read it, going left to right: "Turn any **TAG OFF** where the WORD 'KLATU' is Not Found (NF) on lines currently tagged with :A.

You *could* say, **TAG :A OFF WORD 'KLATU' NF :A**, but since you are *only* dealing with lines *already* tagged as :A and you aren't renaming it to something else, there is no need to specify the target *tag-name* after the TAG command name. It's easier to just leave it out.

A fine point: It will not work if you happen to attempt a command like **TAG :B OFF WORD 'KLATU' NF :A**. In a command like this, all lines selected for the tag command are coming from lines tagged with :A. If you then ask TAG to turn off lines having tag :B, the **TAG OFF** function will confirm the existing tag before

clearing it. It will not find any lines with :B on them, and so you will get the **No lines (re)tagged** message. However, there is nothing wrong in itself with asking for specific tags to be cleared. Suppose you had a range of lines from .ONE to .TWO and you wanted any line tags of :B removed but lines with any *other* tags left alone. You can say **TAG :B OFF .ONE .TWO** and that is perfectly fine.

Now, the results of the tagging operation can be found using the tag :A, and you don't have to keep track of which tag name to use, because there's only one.

One more fine point. The command **TAG OFF 'KLATU' NF :A** removes any tag from lines already tagged with :A in which the string is not found. Suppose you left off the :A tag and just said, **TAG OFF 'KLATU' NF**. Would that "work"? In many cases, it would. What happens is that you would be asking the editor to look at every line of the file, and to remove every tag on every line where *that* particular string is not found. There are two problems with this. First, if the file is very large, re-examining every line, instead of limiting the search to only lines that had *already* been tagged with :A in the prior step, could take longer. Second, if you had any *other* tags you were using for any *other* purpose, they would be cleared out as well. You are free to take this shortcut if you wish, but it's important to understand what you are asking for.

If you get in the habit of doing this type of tagging with a single tag name, you may need to clear out any existing ones before you start. Otherwise your results might include more lines than you intended. Tags can be cleared by using **TAG :A OFF**.

One more thing about **TAG** and **RETRIEVE**. **TAG** is mostly forgiving about the order you type the operands to it. So, the list of commands above can be rewritten to put the string last. Let's do that, and to keep it even simpler, we will dispense with the quotes, too (we are among friends here, right?) So, you'd have this series of commands:

```
TAG :A GORT
TAG OFF NF :A KLATU
TAG OFF NF :A BARADA
TAG OFF NF :A NIKTO
```

Now, for the third and fourth commands, you can retrieve what you entered for the second command, press the End key, backspace over the string (or use the mouse to do the same thing) and then type in the new string each time. That way you only have to type the part of TAG that is different and not redo the whole thing all over again.

Hopefully if you are still with us so far, you may agree that this technique will work and you can follow along the explanation. You may also feel it's a little weird. Remember what we said about **ASSERT** and double negatives? That's why it exists – to address problems like this. The task at hand that we *really* want to do is to *assert* that the desired words are *present*. That particular viewpoint is more important than the whys and hows of turning off tags if the word are *not* present. It is for problems like this that **ASSERT** was made. Let's rewrite this from negative logic to positive logic.

```
TAG :A GORT
TAG ASSERT KLATU :A
TAG ASSERT BARADA :A
TAG ASSERT NIKTO :A
```

Does this make more sense? First, find all lines with GORT, then assert that those lines also have KLATU and the other words. Now, let's take it a step further, and say that we want to make sure (*assert*) that the word REMEMBER is *not* present on any of the lines tagged with :A. That can be done like this:

```
TAG ASSERT REMEMBER NF :A
```

Why go through the trouble of using **ASSERT**? Why was it even made available? Mainly for users editing very large data files. In the process of doing "data mining" activities, they may have a large number of tags set throughout their file. It is very important as one is trimming off tags, looking for exactly the right data, that you ask

for the right things at the right time. By being able to describe tagging operations in the terms (positive or negative logic) that make the most sense to **you** it will decrease the likelihood that you would make a mistake.

The **DROP** and **KEEP** commands

Our discussion of tags so far has dealt with tagged lines as individual 'units' each acted upon independently. When you use **DROP** and **KEEP**, you can deal with tagged lines in groups.

A *tag group* is a set of two or more adjacent lines that have the same tag name. A tag group is preceded and followed by:

- The Top of Data or Bottom of Data line
- A line without a tag
- A line with a different tag than the tag group has

Notice it says *two or more lines*. If you have single, individual tagged lines where the line before or after does not have the same tag, that line is not a tag group. Such 'lone' tagged lines are ignored by **DROP** and **KEEP**.

The **DROP** and **KEEP** may be thought of as "structured **DELETE** commands". The purpose of both of them is to delete lines. The **DROP** command is focused on which lines are to be deleted, while **KEEP** is focused on what lines are *not* to be deleted.

What is the idea behind **DROP** and **KEEP**?

Well, if you have used one or more **TAG** commands to group together blocks of lines having some common feature, it is possible that some of those lines are duplicates. **DROP** and **KEEP** provide a means to quickly get rid of all those duplicate lines in a single command.

DROP and **KEEP** apply their deletion rules to *every tag group in the entire file* having the specified tag name.

The reason there aren't options like **PREV** and **NEXT**, etc. is that if you only wanted to work on a small area, you could just manually delete lines and not use **DROP** and **KEEP**. If you need to work on a large block of the file, but not the entire file, you can use line labels on a **TAG** command to limit their range .

You mean you can have labels *and* tags on the same command at the same time? Yes. It works by selecting lines with the specified tag, but only those within the line label range. This feature works on the **TAG** command, and any others that accept generalized line-range arguments, but not on **DROP** and **KEEP** themselves.

For example, if you had several tag groups defined using tag **:T**, but you didn't want tag groups between lines **.A** and **.B** to be modified, you could use a **TAG** command to temporarily rename the **:T** tags between **.A** and **.B** to something else (let's call it **:TT** for "temporary" **:T** tag). Then, use your **DROP** or **KEEP** as usual, referencing tag **:T**. Once you're done, you can use another **TAG** command to rename the **:TT** tags back to **:T**.

For example, suppose you had a file with blank lines in it, and you wanted to collapse spans of multiple blank lines into single blank lines. Here is a way you could do this:

```
TAG :B OFF
TAG :B P'^' NF
KEEP FIRST :B
```

This does the following:

- Ensure that no lines are tagged with **:B** in the file.
- Search the entire file for lines in which non-blank characters (**P'^'**) are not found (**NF**). Lines that are entirely blank will not have any non-blank characters.
- For each tag block having line tag **:B**, **KEEP** only the **FIRST** line of each block (delete all *other* lines

except for the **FIRST** line of each block)

A command like **TAG :B P'^' NF** may seem a little unusual: Find lines in which *non-blanks are not found*? However, this is necessary to handle situations where a “blank” line is actually a zero-length line. In those cases, there is no data on the line to *find*. So, we have to define the search in terms of *not* finding something, so that those zero-length lines will be found and tagged along with our search for *blank* lines.

This example also illustrates why lone tagged lines are not considered as tag *blocks*. If they were, **DROP** and **KEEP** might delete them as well, and rather than getting rid of *duplicate* lines, you'd get rid of *every* line. There is no need for **DROP** and **KEEP** to delete *every* line, since we already have **DELETE** for that purpose.

Since only *blank* lines were selected, it probably doesn't matter whether the first or last line of each block is retained; the effect is the same. When you apply this technique to your own data, you may have some particular reason why you might want to use the **LAST** line vs. the **FIRST** one.

Recalling the note above, suppose you wanted to remove extra blank lines, but **not** inside the range of **.A** and **.B**, where **.A** precedes **.B** in the file; assume **.A** and **.B** are already defined. **DROP** and **KEEP** don't directly support this, since they do not take generalized line-range operands with label ranges or **X** or **NX** options. Instead, you could accomplish it with something like this:

```
TAG :B OFF
TAG :B P'^' NF
TAG :BT :B .A .B                                -- inside the range .A to .B rename tags :B to :BT
KEEP FIRST :B                                -- the lines tagged with :BT are ignored
TAG :B :BT .A .B                                -- rename the temporary :BT tags back to :B
```

Suppose you had wanted to remove *all* groups of multiple blank lines, but any individual blank lines are to be left as is. This could be done by replacing the **KEEP** command above with **DROP**:

```
DROP ALL :B
```

Using Labels and Tags with the LINE Command

You can manage line labels and line tags using the **LINE** primary command. For example, to put a tag of **:ABC** on line 10, you can issue the command:

```
LINE ':ABC' .10
```

Remember to quote the label, because in this example, the first operand **' :ABC'** is what goes into the sequence area, and the second operand **.10** is where the first operand is placed.

When doing a normal edit, there is usually no need to do such a thing; you would just put the tag on the line directly. However, when running a programmable macro the **LINE** primary command is the only way to do this. You would need to 'wrap' the **LINE** primary command in an **SPF_CMD** function call in your **.MACRO** file, like this:

```
SPF_CMD ("LINE ':ABC' .10")
```

It is possible to set, clear and toggle both line labels and line tags from the **LINE** primary command. The possible options are as follows:

```
LINE '.label'                                -- Enter label into sequence area
LINE '..label'                                -- Toggle label; enter label if no label present,
                                                -- else clear any label that is present
LINE '..'                                    -- Clear any label that may be present
```

LINE ' . '	-- Clear any label that may be present; same as LINE ' . '
LINE ' : tag '	-- Enter tag into sequence area
LINE ' :: tag '	-- Toggle tag; enter tag if no tag present, -- else clear any tag that is present
LINE ' : '	-- Clear any tag that may be present
LINE ' :: '	-- Clear any tag that may be present; same as LINE ' : '

See [LINE - Apply Line Command](#) and [Working with the LINE Primary Command](#) for more information on the LINE command.

See [Special Line Commands](#) in [Using the KEYMAP Dialog](#) for more information on performing these same actions within a KEYMAP definition.

Working with NOTE and xNOTE Lines

Contents of Article

- [Introduction](#)
- [Example of NOTE line command](#)
- [The MD and MN line commands](#)
- [Line commands that can be used on =NOTE> lines](#)
- [Line commands that cannot be used on =NOTE> lines](#)
- [Locating NOTE or xNOTE lines](#)
- [Exporting NOTE or xNOTE lines](#)
- [Importing NOTE or xNOTE lines](#)
- [Managing the exclusion status of NOTE or xNOTE lines](#)
- [Deletion of NOTE or xNOTE lines](#)
- [NOTE lines and the clipboard](#)
- [NOTE lines and highlighting pens](#)

Introduction

SPFLite has the ability to insert **NOTE** lines into data files. A note line is a line of text containing any desired user comments. **NOTE** lines are not part of the data file per se, and do not have line numbers. Other applications that read the data file will see only the data, and not the note lines.

You can also insert **xNOTE** lines. An **xNOTE** is basically the same as a **NOTE**, but you can use any letter from **A** to **Y** to add any of these additional note types into your file, from **ANOTE** to **YNOTE**. It is expected that a primary use of **xNOTE** lines may be in concert with user-written programmable macros, to insert diagnostic lines into an edit file. For example, various **xNOTE** types could be used for varying levels of error severity (such as Advisory, Information, Warning, Error, Severe and Fatal), where a suitable letter like A, I, W, E, S or F would replace the "x" in **xNOTE**. SPFLite does not provide or distribute such macros; you would have to write them yourself.

What can notes be used for? A user could embed notes in a large file to help them keep track of the progress of an extensive editing task. A library of 'models' or 'prototypes' could be created to document common programming tasks. For example, a library of Windows API calls could contain prototypes of calling sequences that included instructions on how to use them; the instructions would be stored as notes. A **NOTE** line could be used as a bookmark, somewhat like a label or tag could be used, but a **NOTE** can provide more information than a label or tag does.

Users of IBM ISPF should be aware the its use of notes is different than in SPFLite. ISPF can have note lines in Dialog Development Models, and can use **NOTE ON** and **NOTE OFF** as primary commands. SPFLite does not implement the **NOTE** primary command, nor Dialog Development Models. However, in SPFLite, if you **COPY** a file containing persistent **NOTE** lines into another file, the **NOTE** lines will be copied as well, an action comparable to copying a Model file in ISPF that contained notes.

See the discussion [NOTE lines and the clipboard](#) below for the considerations that must be taken into account if you want to attempt such a thing.

Notes are stored in the same area that **STATE** information is held (like labels and tags). This means that **STATE ON** must be in effect to store notes. You can create notes with **STATE OFF**, but they will not be retained between sessions.

See [Saving the Edit STATE](#) for more information about STATE processing.

A **NOTE** line is marked by a **=NOTE>** marker in the sequence area.

An **xNOTE** line is marked by an **xNOTE>** marker in the sequence area, where "X" is some letter from A to Y.

If the sequence area width has been reduced to 5, there is not enough room for the full-size **=NOTE>/xNOTE>** markers. Instead, shorter markers are displayed, where **=NOTE>** becomes **=##=>** and **xNOTE>** becomes **=X#=>**.

Any number of note lines may be placed into a file, and they may appear at any desired location(s) within the file.

Types of NOTE lines

SPFLite supports the creation of different *types* of NOTE lines. Creation of different NOTE lines is accomplished by prefixing the **NOTE** command with a letter from A to Y, such as **CNOTE**, **MNOTE** or **XNOTE**.

The xNOTE type **ZNOTE** is reserved; you cannot enter **ZNOTE** as a line command. Instead, **ZNOTE** is used only on the primary command line, to generically reference xNOTE lines of any type (from **ANOTE** to **YNOTE**). **ZNOTE** never refers to "plain" **NOTE** lines, but only the "extended" xNOTE types. You can use the **ZNOTE** keyword on **LOCATE** and **DELETE** primary commands.

New, blank **NOTE** or **xNOTE** lines may be created by the **NOTE** or **xNOTE** line commands, which have the following command syntax:

NOTE*n*
or
xNOTE*n*

The *n* value is used to insert from 1 to 99 blank note lines for **NOTE**, and 1 to 9 blank note lines for **xNOTE**. If omitted or specified as **0** or **00**, one line is inserted. The note line(s) are inserted after the point the **NOTE** or **xNOTE** command is entered on.

If you need more **NOTE** or **xNOTE** lines than the *n* value allows, you can insert one **NOTE/xNOTE** line, and then use the **R** line command on the **NOTE/xNOTE** line with a suitable *n* value to create as many of these lines as you need. Line commands like **R** don't always apply to "special" lines, but for **R** it is permitted.

After the **NOTE** or **xNOTE** line(s) are inserted, they are initially blank. You can move the cursor or mouse into any **NOTE** or **xNOTE** line and edit its content as desired.

NOTE or **xNOTE** lines can be moved, copied and deleted within the edit file. They are ignored for nearly all other purposes.

Because **NOTE** or **xNOTE** lines are not data lines, most primary commands are not applied to **NOTE** or **xNOTE** lines. For example, **FIND** and **CHANGE** do not apply to **NOTE** or **xNOTE** lines.

If it is necessary to find some specific text in a **NOTE** or **xNOTE** line, one approach would be to save the file under a different name, convert all the **NOTE** or **xNOTE** lines into data lines using **MD/MDD**, then do a search of the file for the desired text.

Example of NOTE line command

(before):

```
000010 line ten
NOTE line eleven
000012 line twelve
```

After NOTE line is inserted:

```
000010 line ten
000011 line eleven
=NOTE>
000012 line twelve
```

After NOTE line is updated by user:

```
000010 line ten
000011 line eleven
=NOTE> THIS IS A USER-ENTERED COMMENT LINE
000012 line twelve
```

For special NOTE lines

(before):

```
000010 line ten
MNOTE line eleven
000012 line twelve
```

After MNOTE line is inserted:

```
000010 line ten
000011 line eleven
MNOTE>
000012 line twelve
```

After MNOTE line is updated by user:

```
000010 line ten
000011 line eleven
MNOTE> THIS IS A USER-ENTERED COMMENT LINE
000012 line twelve
```

The MD and MN line commands

The **MD** line command (**MDD** or **MDMD** for blocks) will convert a **=NOTE>** or **xNOTE>** line into a data line.

The **MN** line command (**MNN** or **MNMN** for blocks) will convert a data line into a **=NOTE>** line. In addition, 'special' lines that are displayed by the PROFILE command can also be converted into a normal data line. Such lines include **=PROF>**, **=WORD>**, **=MARK>**, **=TABS>**, **=COLS>** and **=BNDS>**.

Note that the **MN** command always converts lines into "plain" **NOTE>** lines. There is no provision for **MN** to create any type of extended **xNOTE>** lines.

The usual **n** line count and **/** and **** modifiers are permitted.

Example:

Before:

```
000010 line ten
000011 line eleven
=NOTE> THIS IS A USER-ENTERED COMMENT LINE
000012 line twelve
```

MD and MN commands applied:

```
MN      line ten
000011 line eleven
MD      THIS IS A USER-ENTERED COMMENT LINE
000012 line twelve
```

After:

```
=NOTE> line ten
000010 line eleven
000011 THIS IS A USER-ENTERED COMMENT LINE
000012 line twelve
```

Line commands that can be used on =NOTE> or =xNOTE> lines

A
B
BNDS
C/CC
COLS
D/DD
F
H/HH
I
L
LC/LCC, UC/UCC, SC/SCC, TC/TCC
M/MM
MARK
N
R/RR
S/SS
Shift commands
W/WW
WORD
X/XX

Line commands that cannot be used on =NOTE> or =xNOTE> lines

AA
BB
G/GG
J/JJ
O
OR
PL/PLL
T/TT
TB/TBB
TF/TFF
TG/TGG
TJ/TJJ
TL/TLL
TM/TMM
TR/TRR
TS

Locating NOTE or xNOTE lines

The **LOCATE** command has been extended to locate lines of type **NOTE**, so now the command **LOCATE NOTE** is available. The **NOT** option is allowed as well, so non-note lines can be found with **LOCATE NOT NOTE**. All other **LOCATE** features are available as well.

If you are using extended **xNOTE** lines, they can be searched for specifically. e.g. **LOCATE MNOTE** or **LOCATE TNOTE**.

When you use a **LOCATE** command in the form of **LOCATE ZNOTE** it means to generically find **any xNOTE** line of any kind, from **ANOTE** to **YNOTE**. You cannot locate **xNOTE** lines of type **Z** using a keyword of **ZNOTE** because you are not allowed to define such lines. The keyword **ZNOTE** is allowed only on the primary command line, not as a line command.

Exporting NOTE or xNOTE lines

When a file (or any portion thereof) containing note lines is written to disk using **SAVE**, **CREATE** or **REPLACE**, the notes are written as well. For this to happen, the file type of the file being saved must be known to SPFLite; there must be an existing PROFILE for that file type, and that PROFILE must have **STATE ON** enabled. If that is not the case, the file will be saved without the notes; that is, the **NOTE** or **xNOTE** lines will be discarded. When a file has a profile with **STATE ON** in effect, such a file is a *note-enabled file type*. When a file with notes is saved to a note-enabled file type, any note lines are *exported* to the saved file.

Importing NOTE or xNOTE lines

When an external file that has a *note-enabled file type* is copied into a file that is also noted-enabled, and the external file contains **NOTE** or **xNOTE** lines, those **NOTE** or **xNOTE** lines are *imported* into the edit file. The imported **NOTE** or **xNOTE** line(s) retain the same relative position(s) they have in the external file.

Managing the exclusion status of NOTE or xNOTE lines

NOTE and **xNOTE** lines may be made individually excluded or unexcluded using the **X/XX** and **S/SS** line commands. Additionally, you can unexclude all excluded **NOTE** and **xNOTE** lines by using **RESET** or **RESET X**.

Deletion of NOTE or xNOTE lines

A special extension to the **DELETE** command allows for mass-deletion of all note lines. The syntax is:

DELETE NOTE

Note that **DELETE NOTE** does not permit other **DELETE** keywords to be used. In particular, you **cannot** issue a command like **DELETE NOTE ALL** or **DELETE xNOTE ALL**. Within the specified line-control range, or within the entire edit file, **all** notes are deleted; **the "ALL" is implied but cannot be used on these commands.**

The keyword **NOTE** cannot be specified as **NOTES**.

If using special **xNOTE** lines, you may specify a unique note type, like **DELETE CNOTE**, to delete just one type of note, or you can user the reserved **ZNOTE** name. A command of the form **DELETE ZNOTE** will delete all **xNOTE** lines of any kind, from **ANOTE** to **YNOTE**, but will **not** delete "plain" **NOTE** lines. As noted elsewhere, the keyword **ZNOTE** is a generic reference to all **xNOTE** lines; you cannot literally use a line command of **ZNOTE** on a data line.

Note lines may be made individually deleted using the **D/DD** line commands. Presently, there is no means to perform a deletion of **NOTE** lines between line labels, or based on whether they are excluded or not.

It is possible to map a key to find the next **NOTE** line and then delete it. Doing so may help address some of the limitations for **NOTE** deletion. Such a mapped key would be set so that it auto-repeated. For sake of discussion,

suppose we map this function to Ctrl-Shift N. A definition that would accomplish this is:

```
(home) [LOC NOTE] (Enter) (txthome) {D} (Enter)
```

To use this macro to delete a span of **NOTE** lines, the procedure would be:

- Scroll to the first **NOTE** line you want to delete.
- Press and hold Ctrl-Shift N until all desired **NOTE** lines are deleted.

NOTE lines and the clipboard

The SPFLite primary command **CUT** is used to copy data lines into the clipboard, and **PASTE** is used to copy data lines from the clipboard into the edit file. Presently, **CUT** and **PASTE** do not support **NOTE** lines, since there is no portable way to represent **NOTE** lines in the Windows clipboard area while keeping the Windows clipboard in a standard format that is recognized by other application programs.

Still, users may wish to have the ability to copy and paste blocks of lines that have embedded **NOTE** lines. What follows is a suggested approach that may prove useful.

First, establish a temporary file that will be used for copying and pasting lines with embedded notes.

Let us call this file **C:\TEMP\CLIPBOARD.NOTE** for the sake of discussion. The actual file name and file type you use are up to you; this is just an example to explain things.

Next, let us map two keys to copy data to this temporary file and to copy from this file. Some version of the C and V keys could be used, of course, since Ctrl-C and Ctrl-V are used in text editors for a similar purpose. However, in this example, we will use the – minus key for copy and the = equal key for paste, with a modifier key of Ctrl. The mapping will look like this:

```
Ctrl – will contain REP C:\TEMP\CLIPBOARD.NOTE  
Ctrl = will contain COPY C:\TEMP\CLIPBOARD.NOTE
```

When mapping these keys, the check-box for these keys to auto-repeat is cleared, so that they don't unexpectedly repeat themselves.

There MUST be an existing PROFILE for files of type NOTE which has STATE ON enabled. If you have not yet created a profile for files of type NOTE, you will have to make one, and then set its STATE value to ON. That will make the **C:\TEMP\CLIPBOARD.NOTE** temporary file *note-enabled*. **If you omit this step, the technique discussed here WILL NOT WORK.** Bear in mind that having the Profile name be **NOTE** is just a naming convention; you could use any file name or extension you wanted for this Temp file, as long as its Profile has **STATE** set to **ON**.

Once this setup is done, these keys will be sufficient for most purposes. This approach may not work in special cases, such as non-standard file types, fixed-length files or EBCDIC, but for typical ASCII text files, it should work well.

Even though the discussion above references **NOTE** lines, it will also work for **xNOTE** lines. The same technique will work for all types of notes.

NOTE lines and highlighting pens

It is possible to use virtual highlighting pens to color sections of data lines, with the (Pen*) keyboard functions like [\(PenRed\)](#). At present, you cannot use these keyboard functions to color the text of **NOTE** or **xNOTE** lines.

Working with Word and Delimiter Characters

Contents of Article

[*Introduction*](#)

[*Setting the WORD characters*](#)

[*Word Syntax*](#)

Introduction

Many SPFLite functions are required to make decisions based on whether a character is considered to be a Word character or a non-Word (delimiter) character. This support is provided by the [WORD](#) string as part of a file's Profile. This enables the characters that make up a Word to be different for different file types. Example, different programming languages support different character as valid variable names (such as .dot, _ underscore, etc.) IBM mainframe languages often use \$ # and @ as letter characters in names.

In addition, handling international characters which are part of the extended ANSI character set can be a problem. A decision is needed as to whether these should be considered Word characters (as would be normal for western European languages), or should they be non-Word (delimiter) characters, which would typically done for English-based data. This issue is important, because in files containing English data, the presence of international characters is often a symptom of a corrupted file, whereas in a file containing data in French, German or Spanish such characters are normal and expected.

Setting the WORD characters

To set the WORD characters, perform the following:

- Edit a file of the Profile type for which you are setting the WORD values. If needed, go to File Manager and enter **.profile-name** next to the New file option to obtain a dummy edit session.
- Enter OPTIONS as a Primary command. On the General Tab, you will see a setting labeled as "Include international characters for Word/Alphabetic pictures?" Select this if you wish the normal ANSI international word characters to be automatically added to your WORD string. Click DONE when you have made your selection.
- Enter WORD as a line command. The current (default) WORD setting will be displayed as a **=WORD>** line type.
- Enter PROF UNLOCK as a primary command to allow modification of this Profile.
- Edit the **=WORD>** line as needed to suit your Word requirements. See Word Syntax below.
- Your WORD setting is complete

Word Syntax

The WORD string consists of a series of either 1 or 2 byte character strings separated by spaces, or by a character range string separated by spaces. A character range string is simply two characters separated by a - (dash). Two byte character strings are assumed to be Hex character requests and must consist of only valid hex characters 0-9 and A-F, case is unimportant for Hex values. Note that a range can be a character range, or a hex range, but you cannot mix hex and character notation in the same range; so 0-39 and 30-9 are illegal ranges.

The following would be valid operand strings for WORD:

A	the uppercase letter A
FF	the hex value FF (the ý character)
a-z	the whole range of characters from lowercase 'a' through lowercase 'z'
0-9	the numbers '0' through '9'
30-39	the numbers '0' through '9' (expressed in hex)

Range strings must be expressed in ascending sequence. e.g. a range of Z-A is invalid.

The operands of WORD do not have to themselves be in sequence; it is perfectly valid to have a string of:

A-Z FF FE FD a-z _ - 0-9

It is not an error to repeat a character in a WORD specification, the following is valid:

A-Z a-z 0-9 a-c D X z 32-35 a-z

If you have chosen to include the International characters under Options => General as described above, they will be handled as follows. The uppercase International characters will be added if **A-Z** is included in the WORD string; and the lowercase International characters will be added if **a-z** is included in the WORD string.

By automatically adding international characters this way, you don't have to manually list them all yourself (which is a good thing, because there are a lot of them, and they are not in contiguous character ranges either). This "adding" of characters is done internally; you won't visibly see the WORD string modified with the added characters, but SPFLite's processing logic for WORD will take them into account.

Note that the default WORD string used by SPFLite is: **A-Z a-z 0-9**

Use of specific non-English language characters

If you wanted to define only the letters for a specific non-English language (like Spanish) but **not** for other non-English languages, leave the International Characters checkbox disabled, and then manually enter each desired non-English letter you want to include in the list of defined "word letters" on the WORD command.

Why would you want to set up WORD characters that way? Wouldn't it be simpler just to include all international letters and be done with it? Simpler, yes, but not necessarily better. Just as non-English letters in an English-based file could indicate file corruption, non-Spanish letters like **Ð** or **ß** could indicate file corruption in a Spanish-based file. For most user, allowing **all** international letters is far too lenient of a test.

The following describes an approach for doing this.

SPFLite will repopulate the Normal Characters string in the General Options setup dialog when it is deleted. If you enable the International Characters checkbox, clear this field, close the dialog, then reopen it, the Normal Characters field will have every possible displayable character. You could then highlight that string and copy it with Ctrl-C to place the string in the clipboard. Then, paste that string somewhere in an SPFLite edit screen and delete the international characters you didn't want, and reformat that string as a WORD operand. Finally, go back and disable the International Characters checkbox and define your WORD setting as needed.

Use of WORD hex ranges in EBCDIC files

Users of EBCDIC files should be mindful that SPFLite does all of its internal editing in ANSI. EBCDIC files are translated to and from ANSI as needed to make the process appear transparent. What this means for the WORD command is that if you specify a hex range for characters, you must presently use ANSI encoding, not EBCDIC encoding. So, a hex range of digits must be defined as 30-39, even though the data values appear in HEX mode edit displays as F0-F9. As this process could be confusing and non-intuitive, it may be best to avoid hex word ranges for EBCDIC files. Otherwise, choose these values carefully.

This is a limitation in the implementation of the WORD command, which may or may not be addressed in a future

release.

Working with Power Typing Mode

Contents of Article

[Introduction](#)

[Basic concepts of Power Typing](#)

[Power Typing cursor](#)

[Support for LEFT and RIGHT scrolling in Power Typing mode](#)

[Basic features available in Power Typing mode](#)

[Detailed list of keyboard primitive functions allowed in Power Typing mode](#)

[Example 1: Basic features](#)

[Example 2: Right justifying data](#)

[Example 3: Appending a string to a column of varying data](#)

Introduction

Power Typing is an editing facility in which, for each character you type, or each keyboard primitive function you use, the character is typed or the function is applied to every line in the Power Typing line range, in parallel, all at the same time. Every editing action you take is replicated on every line.

Some editors refer to this capability as “column-mode editing” because the same action is applied at the same column position of every line at the same time, in parallel. You will generally find the editing capabilities of SPFLite's Power Typing to be more powerful than the “column mode” features of other editors, because the line range and **X|NX** options allow for the selection of non-contiguous lines, and because of the wide range of keyboard functions you can use while you are within Power Typing mode.

You begin a Power Typing session with the **PTYPE** command, using the lines you specify in the line-range operand or **C/CC** block, which can be limited to just excluded (**X**) or just non-excluded (**NX**) lines if you wish. (See the description of **PTYPE** for more information.)

Once Power Typing mode begins, the edit display moves to the first line of the selected line range, in the same way a **LOCATE** command would do. You will see the message, **Entering Power Type mode, Press Enter to exit**, and the status line will show **PowerType**.

The cursor is then moved to column 1 of that first line, which then acts as a ‘prototype’ or ‘model line’. The concept of the ‘model line’ is especially important for certain aspects of Power Typing, such as highlighting and [enumerations](#).

As you enter characters keys or invoke keyboard functions, you will see the effects reflected on the model line, and on every other line that is included in the Power Typing line range.

Power Typing mode remains in effect until you press Enter. Once that is done, the message **Entering Power Type mode, Press Enter to exit** will disappear, and the status line indicator showing **PowerType** will be removed as well.

Note 1: Power Typing mode is allowed while within a multi-edit session.

Note 2: Keyboard Recording is allowed during Power Typing. However, since the status indicator is already showing **PowerType**, the usual message of **KB Recording** will not appear. However, keyboard recording will still proceed as usual. When you press the key mapped to **(Record)**, which by default is the Scroll Lock key, you will see the SPFLite KeyMap dialog will appear. When you exit from KeyMap, Power Typing continues in effect. You can immediately use any keys you have added or changed in KeyMap in your resumed Power Typing session.

Basic concepts of Power Typing

When you are Power Typing, conceptually you are typing on one line – the top line of the line range you selected when you issued the **PTYPE** command. This top line is called the **prototype** or **model line**. As you type on or edit the model line, every character you type and every editing action you take is duplicated on every other line in the line range at the same time, in parallel.

Since the design of Power Typing is based on the 'model line' approach, you are constrained to focus your editing activities on this first 'model line' even though the changes you make to it are propagated to every other line you are working with. Because of this, when you are in Power Typing mode, certain keystrokes, commands and editing features are not available:

- No line commands
- No primary commands
- No page **UP** or page **DOWN** scrolling functions
- No [\(Up\)](#) or [\(Down\)](#) cursor movement functions
- No cursor movements that would leave the current line, like [\(Home\)](#)

Because you cannot use line command or primary commands, much of the power of Power Typing comes from the extensive list of keyboard primitive functions that are allowed in Power Typing mode, which include almost all of the functions allowed during ordinary editing. If any of those functions provide editing you might wish to use during Power Typing, be sure to KEYMAP them to key combinations of your choice, so you'll have them at hand and ready to use when you begin a Power Typing session.

Note: Because the keyboard functions are a major source of Power Typing's **power**, that's a good reason why you should invest the time to understand how keyboard functions and key mapping operate, to get the most benefit out of Power Typing. The dividends you'll receive by doing so will make the learning curve is well worth it.

See the Power Typing examples at the end of this article for more information.

Power Typing cursor

When the **PTYPE** command is issued, there is an implied multi-line cursor in effect, since all editing actions that take place on the top (model) line are replicated on every line that is within the **PTYPE** line range. A set of vertical lines, one on each side of the real cursor will appear, starting from the real cursor itself to the last line of the **PTYPE** line range. These lines are displayed using the same graphics used for MARK lines, and in the selected MARK color. This makes the point of editing activity on each line plainly visible. All of the screen shots in this section have been revised to show these new multi-line cursors.

Support for LEFT and RIGHT scrolling in Power Typing mode

If you wish to scroll the file left or right while Power Typing, you can use the left and right arrows. Even though SPFLite in general does not allow primary commands to be executed while Power Typing, special support has been added so you can use keys that are mapped to the primary commands **LEFT** and **RIGHT** to accomplish this. Most users will have these keys mapped to F10 for **LEFT** and F11 for **RIGHT**. You cannot enter these actual commands while Power Typing, because you cannot move the cursor to the primary command area. But, you can type a mapped key, like F10 or F11, to do this. When this is done, SPFLite will respect the current **Scroll** amount displayed in the upper-right corner of the screen.

Basic features available in Power Typing mode

You can do the following basic activities in Power Typing mode, assuming you have typical key mappings set up, most of which you will already have available by default:

- Enter ordinary text, including special characters assigned in KEYMAP

- Move the cursor left and right
- Move the cursor to the beginning or end of the line
- Delete characters with DEL or Backspace key
- Erase to end of line
- Highlight text
- Delete highlighted text via the DEL or Backspace key
- Replace highlighted text with new text, with or without the INS mode being on

You can also justify highlighted text (left, right or centered) and [enumerate](#) highlighted text as decimal or hex numbers. These functions do not have default key mappings.

Detailed list of keyboard primitive functions allowed in Power Typing mode

You can use the following keyboard functions while you are in Power Typing mode. A brief description of the effect of each function is provided. Any keyboard function that involves a clipboard can reference a Named Private Clipboard. For example, if you have a Named Private Clipboard of **myclip**, you can paste data from it during a Power Typing session with a function of **(Paste / myclip)**

(Backspace)

Moves cursor left one column on the top (model) line, then deletes the character at that position on every line.

(BackTab)

Moves cursor to prior tab position on the top (model) line.

(ClipDate)

Stores date string, in Windows-defined format, into the clipboard.

(ClipIsoDate)

Stores date string, in format YYYY-MM-DD, into the clipboard.

(ClipIsoTime)

Stores time string, in format HH:MM:SS, into the clipboard.

(ClipName)

Stores the basic file name of the edit file into the clipboard.

(ClipPath)

Stores the fully-qua file name of the edit file into the clipboard.

(ClipTime)

Stores time string, in Windows-defined format, into the clipboard.

(Copy)

Copy highlighted text from the top (model) line into the clipboard.

(CopyPaste)

If there is currently highlighted text on the top (model) line, it performs a (Copy) operation. If there is no current highlighted text, it performs a (Paste) operation, storing the pasted text on every line beginning at the column where the cursor is located on the top (model) line.

(Cut)

If there is currently highlighted text on the top (model) line, this will copy it to the Clipboard and will delete the selected characters from the top line, and from the same corresponding positions on every line. For example, if the highlighted text is in columns 1-5 on the top line, the text going to the clipboard comes from columns 1-5 of the top line, and columns 1-5 are deleted from every line. Characters to the right on each line are shifted left.

(Date)

The current date, in Windows-defined format, is stored into every line at the current cursor location.

(Delete)

The character at the current cursor location is deleted from every line. If there is currently highlighted text on the top (model) line, characters in the same corresponding positions are deleted on every line.

(EndOfLine)

Move the cursor to 1 character past the current end of the top (model) line. If the top (model) line contains trailing blanks, the cursor is placed after the last blank of that line.

(EndOfText)

Move the cursor to 1 character past the last non-blank character in the top (model) line.

(Enum)

Enumerate (create sequence numbers) on each line of the Power Typing line range. There must be a highlighted field on the top (model) line, which holds the starting value and format information. (Enum) creates sequence numbers in decimal. To successfully enumerate a range of lines, Power Typing must be active, a highlighted field must be present, and an initial value with optional formatting, are required. See [Working With Enumerations](#) for more information.

(EnumHexLc)

Enumerate (create sequence numbers) on each line of the Power Typing line range. There must be a highlighted field on the top (model) line, which holds the starting value and format information. (EnumHexLc) creates sequence numbers in hex, and where hex numbers contain digits greater than **9**, the digits will be formatted as **a-f** in Lower Case. To successfully enumerate a range of lines, Power Typing must be active, a highlighted field must be present, and an initial value with optional formatting, are required. See [Working With Enumerations](#) for more information.

(EnumHexUc)

Enumerate (create sequence numbers) on each line of the Power Typing line range. There must be a highlighted field on the top (model) line, which holds the starting value and format information. (EnumHexUc) creates sequence numbers in hex, and where hex numbers contain digits greater than **9**, the digits will be formatted as **A-F** in Upper Case. To successfully enumerate a range of lines, Power Typing must be active, a highlighted field must be present, and an initial value with optional formatting, are required. See [Working With Enumerations](#) for more information.

(Erase)

Will replace all highlighted characters with an equal number of spaces, on the top (model) line, and every other line in the the Power Typing range in the same columns. It is an error to attempt to use (Erase) if no highlighting is present.

(EraseEol)

Will erase (delete) all characters from the cursor location to the end of line, including the character at the cursor location, on every line.

(Insert)

Will toggle the current Insert/Overtype status. The current status is always displayed in the status line.

(IsoDate)

Will paste the current date, in ISO format, into the text at the current cursor location, on every line. ISO date format is YYYY-MM-DD.

(IsoTime)

Will paste the current time, in ISO format, into the text at the current cursor location, on every line. ISO time format is HH:MM:SS, 24 hr clock.

(JustifyC)

Text justification requires highlighted text on the top (model) line. The highlighted field defines a column range used during the text justification. (JustifyC) will center the text in the middle of the column range, on every line. Where an odd number of characters are involved in centering, there will be one more character on the right side of center than on the left side of center.

(JustifyL)

Text justification requires highlighted text on the top (model) line. The highlighted field defines a column range used during the text justification. (JustifyL) will left-justify the text within the column range, on every line.

(JustifyR)

Text justification requires highlighted text on the top (model) line. The highlighted field defines a column range used during the text justification. (JustifyR) will right-justify the text within the column range, on every line.

(LastTab)

Will move the cursor to the last defined tab in the Tabs line, if Tabs are currently active. If the last tab code on the Tabs line is a + plus sign, that is where the cursor will be positioned.

(Left)

Will move the cursor one character to the left.

(Lift)

Will copy all highlighted characters into the clipboard and then replace them with an equal number of spaces. Thus, the function is used to "lift" characters off the screen without the surrounding characters moving in any way. It is an error to attempt to use (Lift) if no characters are currently highlighted.

(LowerCase)

If there is currently highlighted text on the top (model) line, the highlighted field defines a column range used during the operation. (LowerCase) will convert text to lower case within the column range, on every line.

(MarkEnd)

Will move the cursor to the last text character on the top (model) line in text selection mode. If selection mode is not already set, it will turn on mark mode and highlight the text from the current cursor location to the last text character on the top (model) line, and on the same corresponding columns on every line.

(MarkLeft)

Will move the cursor left one character on the top (model) line in text selection mode. If selection mode is not already set, it will turn on mark mode and highlight the current cursor location on the top (model) line, and on the same corresponding columns on every line.

(MarkRight)

Will move the cursor right one character on the top (model) line in text selection mode. If mark selection is not already set, it will turn on mark mode and highlight the current cursor location on the top (model) line, and on the same corresponding columns on every line.

(Paste)

Will paste the current Clipboard contents at the current cursor location, and on the same corresponding columns on every line. If the Clipboard contains multiple text lines, only the first line in the clipboard is used. The same string value is pasted into every line.

(PowerCopy)

There must be currently highlighted text on the top (model) line. The highlighted field defines a column range used during the copy operation. The range of columns are copied to the clipboard from every line, as an array of individual string values. The format of the clipboard data is one text line per field copied, one

per line in the Power Typing line range. That means you can paste the clipboard data from (PowerCopy), as a list of lines, into any Windows application using standard paste commands, such as Ctrl-V in NotePad.

(PowerCut)

This function operates the same way as (PowerCopy), except that the characters in the column range are deleted after being copied into the clipboard.

(PowerPaste)

The function requires that the clipboard contain one or more lines of text; these lines need not be of the same length. The lines of text are pasted left-justified starting at the column where the cursor is on the top (model) line. If there are more lines in the clipboard than lines in the Power Typing line range, the extra lines in the clipboard are not used. If there are more lines in the Power Typing line range than lines in the clipboard, the extra lines in the Power Typing line range are unchanged. If Insert Mode is on, data for each line in the clipboard pushes over the existing data on each line, and this data is inserted one line at a time, even when the lines in the clipboard are of differing lengths.

(ResetInsert)

Saves the current setting of Insert Mode (for possible later use by RestoreInsert), and then turns Insert Mode off.

(RestoreCursor)

Will restore the column position of the cursor from a prior (**SaveCursor**) function. When (**SaveCursor**) and (**RestoreCursor**) are using in Power Typing mode, they should be used together. No attempt should be made to save the cursor outside of Power Typing mode and then restore it while Power Typing is active, or vice-versa.

(RestoreInsert)

Sets the Insert Mode to whatever it was (on or off) before the most recent (SetInsert) or (ResetInsert) was done. If neither function has been done since SPFLite was started, (RestoreInsert) has no effect.

(Right)

Will move the cursor one character to the right.

(SaveCursor)

Will save the column position of the cursor, to be restored later by a (**RestoreCursor**) function.

(SentenceCase)

If there is currently highlighted text on the top (model) line, the highlighted field defines a column range used during the operation. (**SentenceCase**) will convert the first letter of the first word to upper case within the column range, and all other letters are converted to lower case, on every line.

(SetInsert)

Saves the current setting of Insert Mode (for possible later use by RestoreInsert), and then turns Insert Mode on.

(ScrollLeft)

Scrolls the screen to the left. The cursor remains in its current location.

(ScrollRight)

Scrolls the screen to the right. The cursor remains in its current location.

(Swap)

Will swap two areas marked on the model line in each PowerType line.

(Tab)

Will move the cursor to the next tab stop if in the text area and Tabs are active. In Power Typing mode,

(Tab) will never move the cursor off the top (model) line.

(Time)

Will paste the current time into the text at the current cursor location, on every line. Format is HH:MM:SS AM. 12 hour clock with trailing AM/PM.

(TitleCase)

If there is currently highlighted text on the top (model) line, the highlighted field defines a column range used during the operation. (TitleCase) will convert text to title case (capitalizing the first letter of every word, and lower-casing the rest of each word) within the column range, on every line.

(ToggleHome)

Based upon the contents of the top (model) line, if the cursor is at column 1, it will get repositioned to the left-most non-blank of the line; if not at column 1, it will get repositioned to column 1; if the line is blank, it will always position the cursor to column 1.

(TxtHome)

Will move the cursor to column 1 of the text data.

(UpperCase)

If there is currently highlighted text on the top (model) line, the highlighted field defines a column range used during the operation. (UpperCase) will convert text to upper case within the column range, on every line.

(WordLeft)

Will move the cursor left to the beginning of the current word (if within a word) or to the beginning of the previous word if already at the beginning of a word. Cursor will never move off the current line.

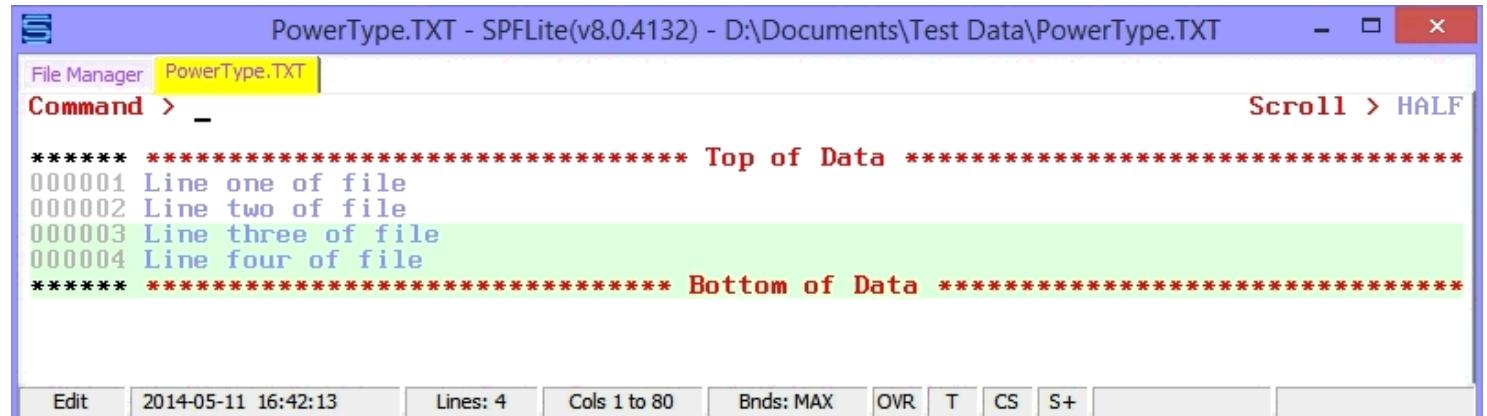
(WordRight)

Will move the cursor left to the beginning of the next word. Cursor will never move off the current line.

Example 1: Basic features

Because Power Typing is a very dynamic process, it can be hard to capture the 'feel' of it in a static Help document such as this, but we will do our best.

Let's start with a small file:

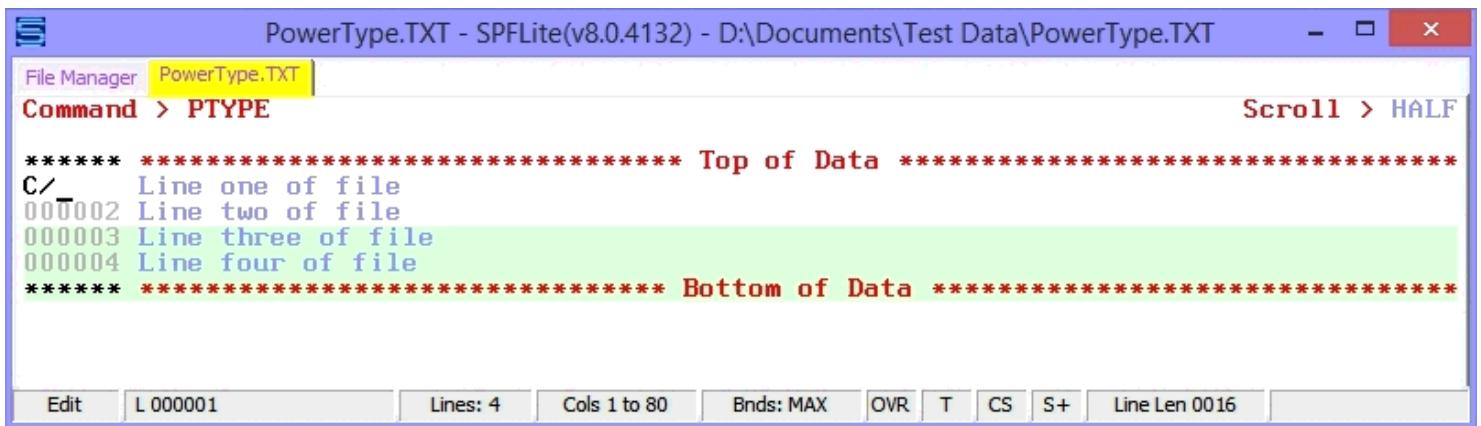


The screenshot shows a window titled "PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT". The file content is as follows:

```
***** * ***** * ***** * ***** * Top of Data * ***** * ***** * ***** *  
000001 Line one of file  
000002 Line two of file  
000003 Line three of file  
000004 Line four of file  
***** * ***** * ***** * Bottom of Data * ***** * ***** * ***** *
```

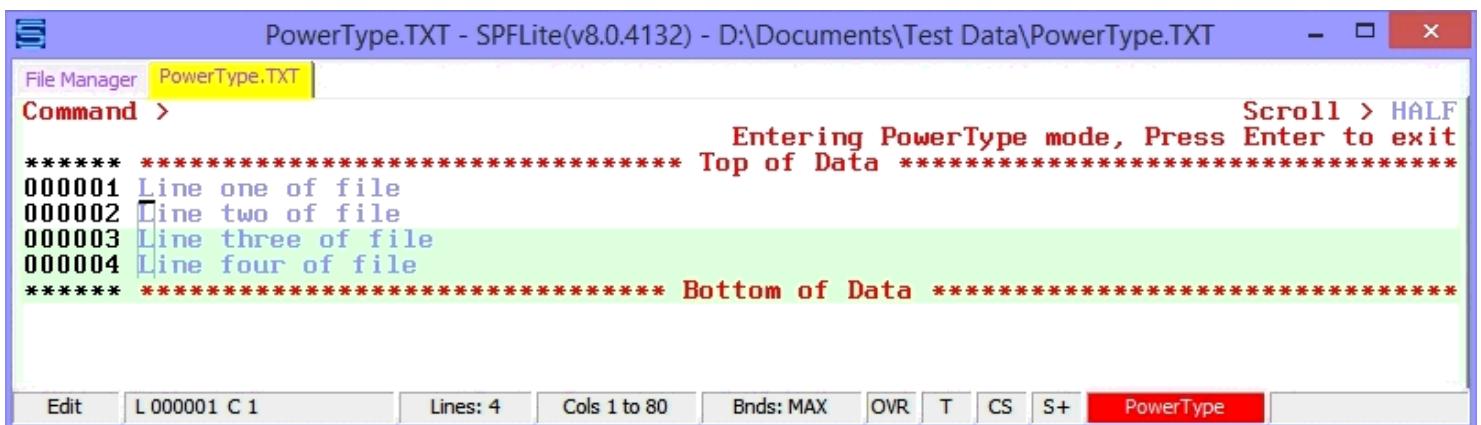
The text "Top of Data" is in red, and the text "Bottom of Data" is in green. The file status bar at the bottom shows: Edit, 2014-05-11 16:42:13, Lines: 4, Cols 1 to 80, Bnds: MAX, OVR, T, CS, S+, and several empty buttons.

There are a number of ways of starting Power Typing mode on this file. Here is one using a **C/** line command:



```
***** **** Top of Data ****
C/_ Line one of file
000002 Line two of file
000003 Line three of file
000004 Line four of file
***** **** Bottom of Data ****
```

After you press Enter, you will see the Power Typing message and the **PowerType** indicator on the status line:



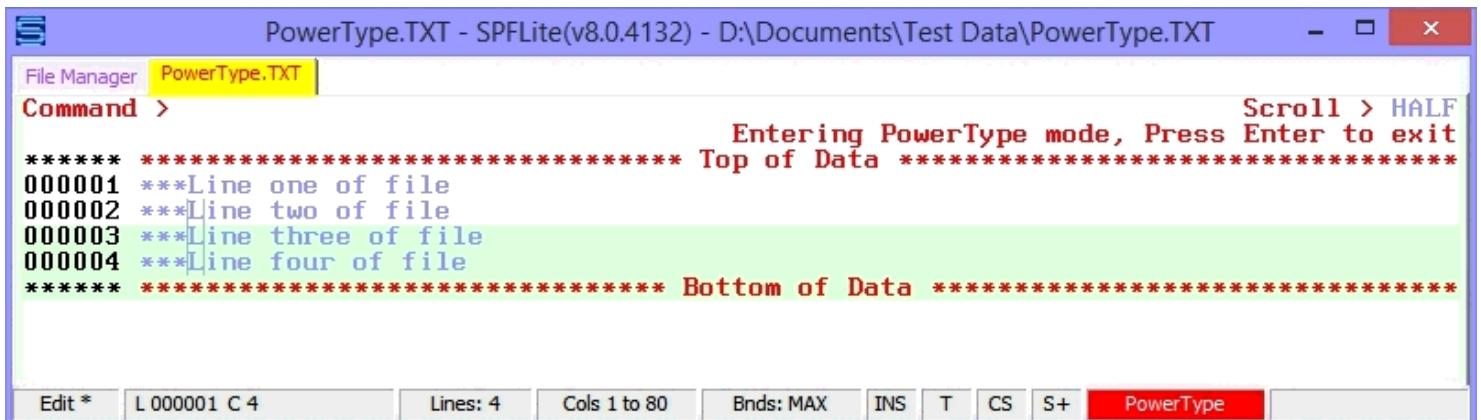
```
***** **** Top of Data ****
000001 Line one of file
000002 Line two of file
000003 Line three of file
000004 Line four of file
***** **** Bottom of Data ****
```

Notice that all the data lines are now highlighted. That tells you which lines are involved in the Power Typing session. This issue becomes more important if those lines are non-contiguous, such as if less than the whole file is involved, if you said X or NX on the PTYPE command, or if you used tags to select a non-contiguous set of lines, etc. Since we chose the whole file, the top (model) line is line 1.

Notice also the cursor is at column 1 of line 1, and the implied cursor is visibly shown with the vertical lines extending downward from the real cursor. That is the point where all data is going to be entered on every line.

Keep in mind that you will stay in Power Typing mode until you press the Enter key.

When you enter Power Typing mode, you will be placed in Overtype mode (OVR will appear on the status line) *even if it had been INS mode prior to starting PTYPE mode*. Now, set Insert Mode on, and type the * key three times. You will see this:



```
***** **** Top of Data ****
000001 ***Line one of file
000002 ***Line two of file
000003 ***Line three of file
000004 ***Line four of file
***** **** Bottom of Data ****
```

The *** is inserted into every line at the same time. Now, turn Insert Mode off. Suppose we want to change the

lines so that the “one, two, three, four” is replaced by a sequential 3-digit number (an enumeration) starting with **501**. First, move the cursor to column 9 and press the key mapped to Erase to End of Line, normally mapped to the ESC key. You will see all the text from column 9 onward is erased on every line:

```
***** *****
000001 ***Line
000002 ***Line
000003 ***Line
000004 ***Line
***** *****

```

Entering PowerType mode, Press Enter to exit

Edit * L 000001 C 9 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType

Next, replace the former variable-length text with a fixed-length string that includes the starting number.

You will type in the string, **501 of the file**. You type this string just once, and as you do, the text gets replicated on every line, at the same time:

```
***** *****
000001 ***Line 501 of the file
000002 ***Line 501 of the file
000003 ***Line 501 of the file
000004 ***Line 501 of the file
***** *****

```

Entering PowerType mode, Press Enter to exit

Edit * L 000001 C 24 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType

Now, highlight the number 501 on the first line (as you will recall, the first line is the only one you can operate on). You can use the mouse or the keyboard (using arrow keys mapped to ‘mark’ keyboard functions) to do the highlighting.

```
***** *****
000001 ***Line 501 of the file
000002 ***Line 501 of the file
000003 ***Line 501 of the file
000004 ***Line 501 of the file
***** *****

```

Entering PowerType mode, Press Enter to exit

Edit * L 000001 C 11 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PT Len 3 @ 9 11 # .1.1

To complete the enumeration, you would type a key mapped to the [\(Enum\)](#) function. Suppose you mapped Ctrl-E to do this prior to beginning your Power Typing session. Assuming you have, type Ctrl-E now. The enumeration will be done on every line, at the same time, and you will see this:

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT

File Manager PowerType.TXT

Command > Scroll > HALF
Entering PowerType mode, Press Enter to exit

```
***** **** Line 501 of the file
000001 *** Line 502 of the file
000002 *** Line 503 of the file
000003 *** Line 504 of the file
***** **** Bottom of Data *****
```

Edit * L 000001 C 11 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 9 11 # .1.1

Example 2: Right justifying data

As another example, let's go back to the original data:

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT

File Manager PowerType.TXT

Command > Scroll > HALF
Entering PowerType mode, Press Enter to exit

```
***** **** Line one of file
000001 Line two of file
000002 Line three of file
000003 Line four of file
***** **** Bottom of Data *****
```

Edit * L 000001 C 1 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 9 11 # .1.1

We will highlight columns 6 through 18 to cover the variable-length part of these lines:

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT

File Manager PowerType.TXT

Command > Scroll > HALF
Entering PowerType mode, Press Enter to exit

```
***** **** Line one of file
000001 Line two of file
000002 Line three of file
000003 Line four of file
***** **** Bottom of Data *****
```

Edit * L 000001 C 18 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PT Len 13 @ 6 18 # .1.1

For sake of discussion, we will assume that the Justify functions are assigned as:

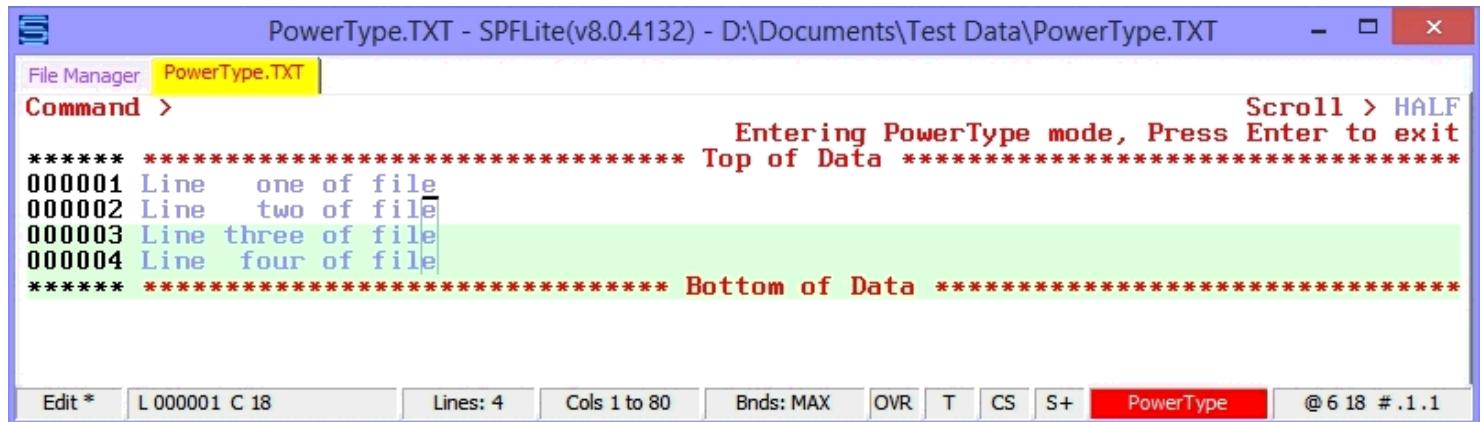
Ctrl-Shift [= [\(JustifyL\)](#)
 Ctrl-Shift] = [\(JustifyR\)](#)
 Ctrl-Shift \ = [\(JustifyC\)](#)

There are no installation defaults for the Justify functions, but you can make these anything you wish, of course.

Assuming you have your keys mapped that way, press **Ctrl Shift]** now.

This will right-justify the text on every line, in columns 6 through 18. Once done, you will see the text justified on

each line:



PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT

File Manager PowerType.TXT

Command > Scroll > HALF
Entering PowerType mode, Press Enter to exit

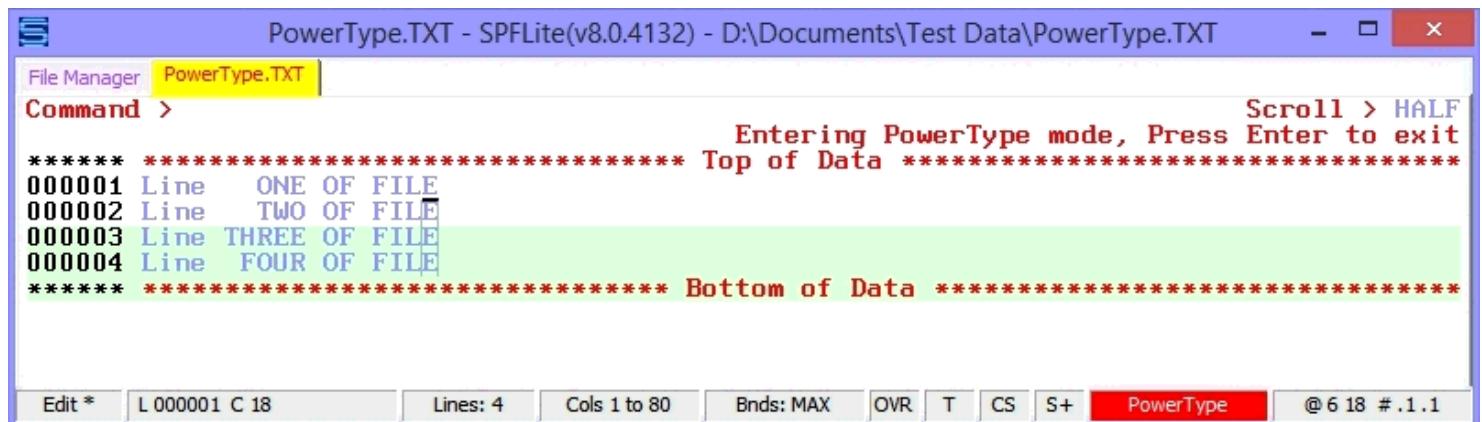
***** **** Top of Data *****

000001 Line one of file
000002 Line two of file
000003 Line three of file
000004 Line four of file

***** Bottom of Data *****

Edit * L 000001 C 18 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 6 18 # .1.1

If you highlighted the same columns and pressed a key mapped to [\(UpperCase\)](#) it would convert those columns to upper case on every line, at the same time:



PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT

File Manager PowerType.TXT

Command > Scroll > HALF
Entering PowerType mode, Press Enter to exit

***** **** Top of Data *****

000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE

***** Bottom of Data *****

Edit * L 000001 C 18 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 6 18 # .1.1

These are just a few examples of what is possible with Power Typing. You will also find the Power Copy, Power Cut and Power Paste functions to be especially useful. Once you have finished, just press the Enter key, and you will return to normal editing.

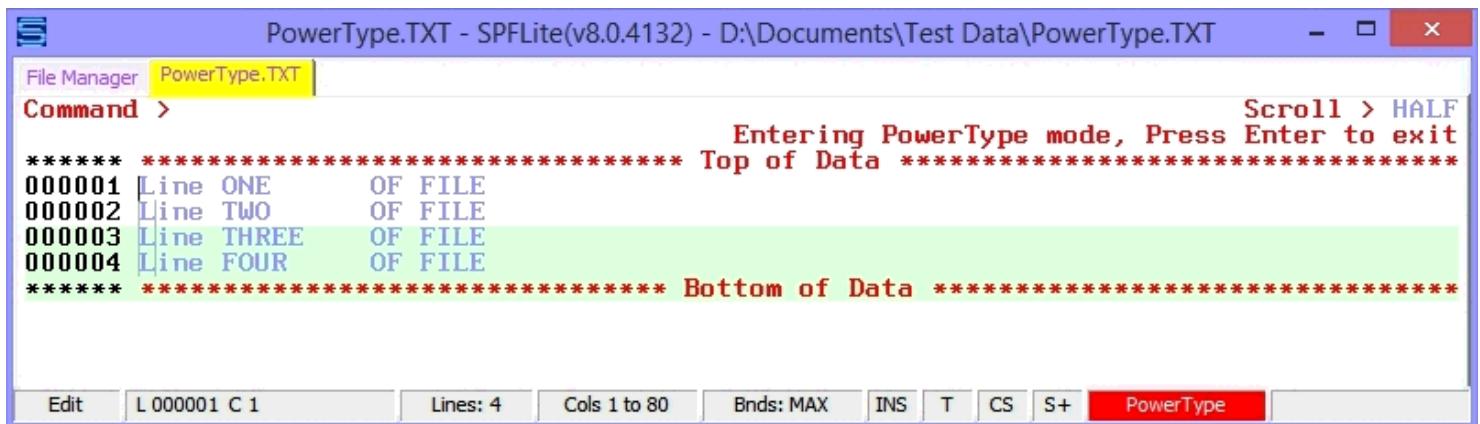
Example 3: Appending a string to a column of varying data

The [APPEND](#) primary command can be used to append a string to the end of multiple lines of varying lengths. For example, you could put a period at the end of a group of sentences between labels **.A** and **.B** by the command **APPEND ' .' .A .B ALL**. But suppose you had a column of values and you needed to add some value to the end of each item, but the items were in the middle of the line rather than at the end. How could you do it? **Assuming all of the column of values are left-justified to begin with**, you can use Power Typing to achieve this.

To do this, you will need to map the text-justification functions to keys to make them available. For sake of discussion, let's again assume the following key mappings are in place. As usual, these are not defaults, and you would have to map these keys yourself.

Ctrl-Shift [= [\(JustifyL\)](#)
Ctrl-Shift] = [\(JustifyR\)](#)
Ctrl-Shift \ = [\(JustifyC\)](#)

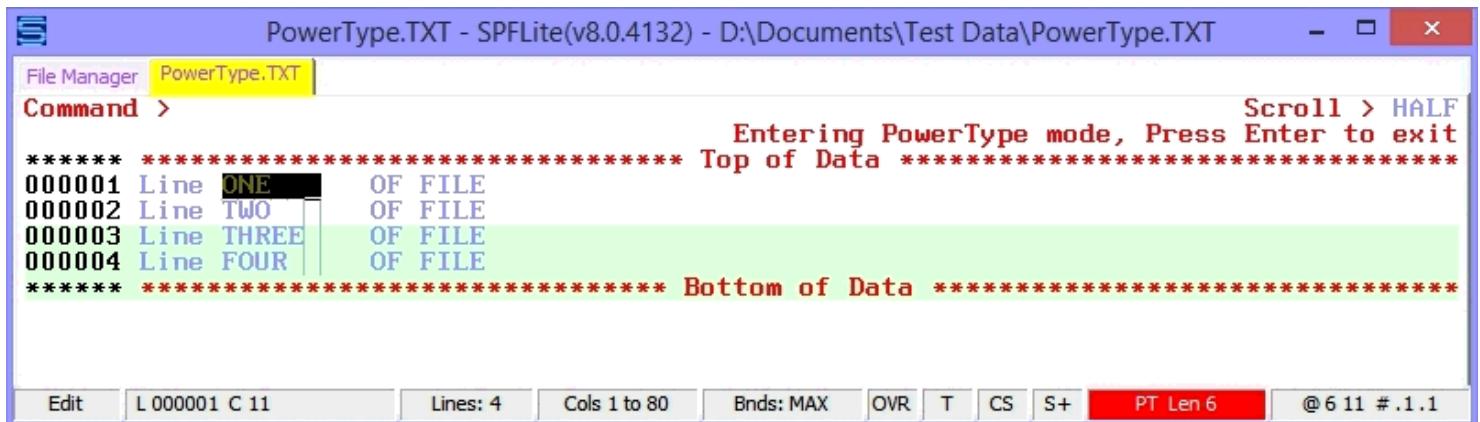
Now, let's make a file, similar to the one above, with some data we need to modify, and we will enter Power Typing mode:



```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command > Scroll > HALF
***** **** Top of Data **** Bottom of Data ****
000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE
***** Bottom of Data ****
Edit L 000001 C 1 Lines: 4 Cols 1 to 80 Bnds: MAX INS T CS S+ PowerType PT Len 6
```

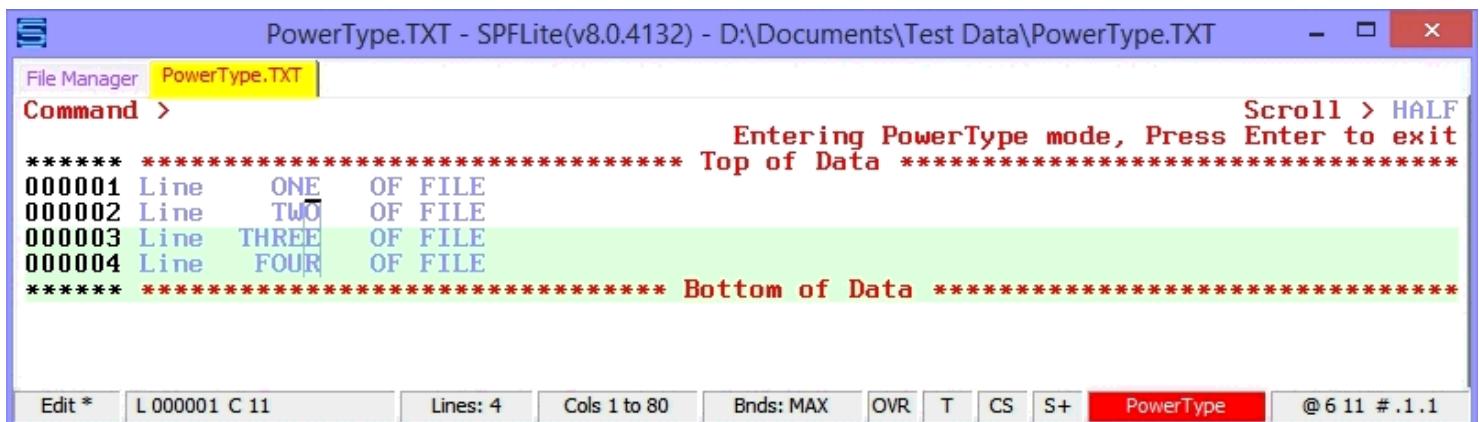
Suppose the task is to put parentheses around each of the words ONE, TWO, THREE and FOUR. Since these words are of varying lengths, how do we do it? First, highlight the columns over the words, so that the highlighting covers the longest word, which in this case is the five letters of the word THREE, and then one extra column. Notice that the status line shows PT Len 6, an indication that you are still in Power Typing mode and you highlighted a string of length 6.

This gives us:



```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command > Scroll > HALF
***** **** Top of Data **** Bottom of Data ****
000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE
***** Bottom of Data ****
Edit L 000001 C 11 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PT Len 6 @ 6 11 # .1.1
```

Now, noting the keys you mapped above, right-justify the column of text:



```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command > Scroll > HALF
***** **** Top of Data **** Bottom of Data ****
000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE
***** Bottom of Data ****
Edit * L 000001 C 11 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType PT Len 6 @ 6 11 # .1.1
```

Move the cursor to column 12 and type a right parenthesis:

Highlight the columns again as shown here:

Again, noting the keys you mapped above, left-justify the column of text:

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT - - □ ×

File Manager PowerType.TXT

Command > Scroll > HALF

Entering PowerType mode, Press Enter to exit

***** **** Top of Data *****

000001 Line ONE) OF FILE
000002 Line TWO) OF FILE
000003 Line THREE) OF FILE
000004 Line FOUR) OF FILE

***** Bottom of Data *****

Edit * L 000001 C 7 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 7 12 # .1.1

Finally, move the cursor to column 6, type a left parenthesis, then press Enter to end Power Typing mode. Result:



File Manager PowerType.TXT

Command >

Scroll > HALF

***** ***** Top of Data *****
000001 Line (ONE) OF FILE
000002 Line (TWO) OF FILE
000003 Line (THREE) OF FILE
000004 Line (FOUR) OF FILE
***** ***** Bottom of Data *****

Edit *

2014-05-12 13:50:44

Lines: 4

Cols 1 to 80

Bnds: MAX

OVR

T

CS

S+

@ 7 12 # .1 .1

Working with Enumerations

Contents of Article

[Introduction](#)

[Combining Fast Enumeration with Line Exclusion](#)

[Enumeration patterns](#)

[Examples](#)

[Handling of blanks in the prefix](#)

[Use of the ENUMWITH command](#)

[Application Note: Using a keyboard macro to automatically insert sequence numbers](#)

Introduction

The Enumerate Facility allows a selected set of columns in a range of lines to be **enumerated**, that is, sequenced starting from an initial value on the first line and incremented for each subsequent line, within a fixed field on a set of lines. Enumeration values are always right-justified within the field. The increment amount defaults to **1** and can be changed with the [ENUMWITH](#) primary command; see discussion below.

There are two kinds of enumeration that you may perform:

Power Enumeration

- Power Typing mode must be enabled
- One or more columns on the first line (the 'model line' of the Power Typing line range) must be highlighted
- The first highlighted line must contain a decimal number in the highlighted field if you intend to use [\(Enum\)](#), or a hex number in the field if you intend to use [\(EnumHexLc\)](#) or [\(EnumHexUc\)](#).

Fast Enumeration

- A multiple-line ("2-D") column of text is highlighted (Power Typing is not in effect)
- The first highlighted line must contain a decimal number in the highlighted field if you intend to use [\(Enum\)](#), or a hex number in the field if you intend to use [\(EnumHexLc\)](#) or [\(EnumHexUc\)](#).

When these conditions are true, a mapped Enumerate keyboard primitive command can be issued. If the requisite conditions are not present, attempting these commands will result in an error message. The Enumerate commands that you can map to a key are the following:

- [\(Enum\)](#)
- [\(EnumHexLc\)](#)
- [\(EnumHexUc\)](#)

The two hex enumeration functions control how digits A-F should be cased, where [\(EnumHexLc\)](#) makes digits greater than 9 in lower case, and [\(EnumHexUc\)](#) makes them upper case. Based on user preference, it is likely that a given user would only select one of these two functions on a regular basis.

The highlighted field on the first line of selected text constitute the '*initial*' or '*base*' value of the enumeration, defining the first value in the sequence. All subsequent lines (in increasing line number order) in the line range are incremented for each new line.

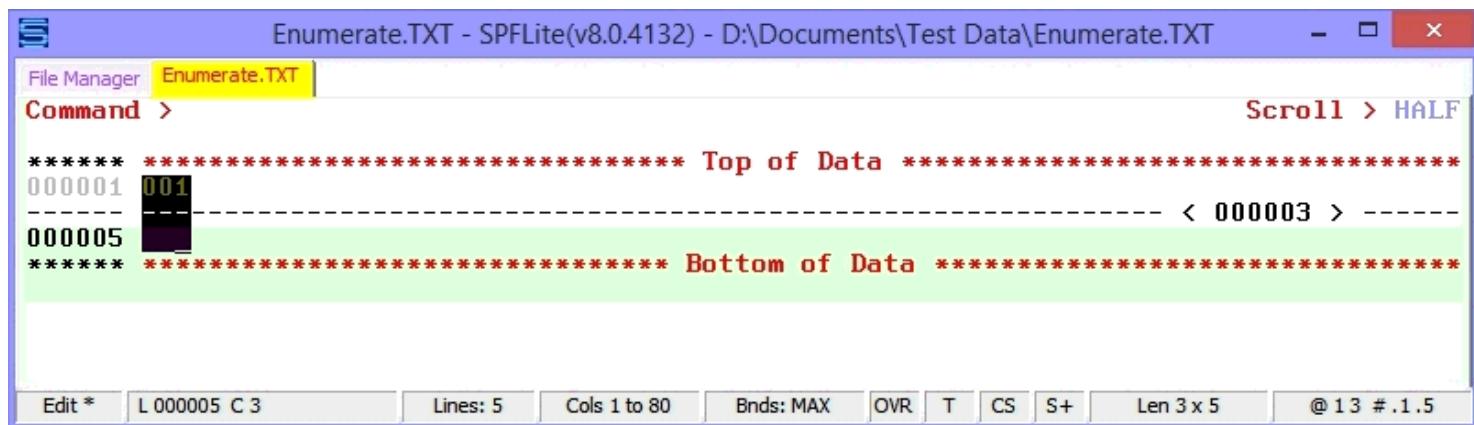
If a Power Typing range is being used, it might not consist of physically adjacent lines, such as when **X** | **NX** or **U** | **NU** is specified, or a range of tagged lines is specified. The enumeration only applies to the *selected* lines, and has no bearing on the *physical* line numbering involved. So if physical lines 1 and 7 are being 'Power Typed' (but

(not the lines between them) and an enumeration starts at line 1 begins with a base value of **5001** and an increment of **1**, line 7 gets the *second* enumeration value of **5002**, *not* the seventh one which would have been **5007**.

Combining Fast Enumeration with Line Exclusion

If you would like to quickly enumerate many lines, but also want to convenience of the Fast Enumeration method, you can exclude the majority of the affected lines, leaving only the first and last ones visible. Then, set up your model line, and enumerate the range. You will see the full range enumerated once you unexclude them.

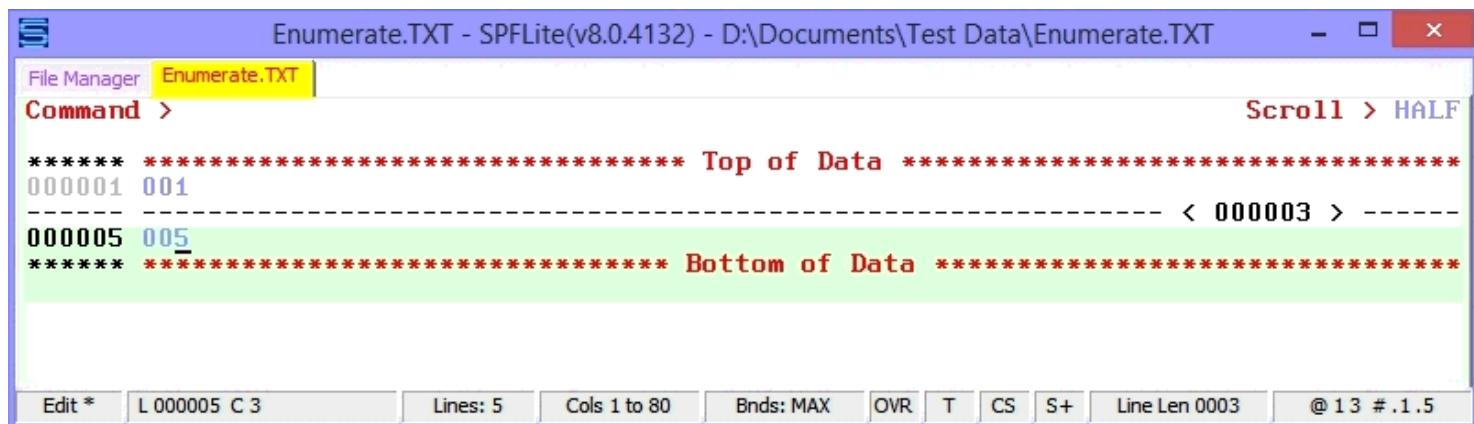
Example: Lines 1 to 5 need to be enumerated. (The example is small, but imagine if you had hundreds or thousands of lines; the same method would still work.) Lines 2-4 are excluded, line 1 is set up as the model line, a starting value is created, and then the entire line range is highlighted. This works because the text-selection highlighting is also applied to all the lines within the the excluded range. Notice on the status line the size of the selected area is a length of 3 characters wide by 5 lines (**Len 3 x 5**), even though you only physically selected 3 lines.



***** * ***** Top of Data *****
000001 001
----- < 000003 > -----
000005 ***** Bottom of Data *****

***** * ***** Top of Data *****
Edit * L 000005 C 3 Lines: 5 Cols 1 to 80 Bnds: MAX OVR T CS S+ Len 3 x 5 @ 13 # .1.5

Then, issue the desired enumeration function as you have mapped it. You will see the final line containing 005 as expected:



***** * ***** Top of Data *****
000001 001
----- < 000003 > -----
000005 005 ***** Bottom of Data *****

***** * ***** Top of Data *****
Edit * L 000005 C 3 Lines: 5 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003 @ 13 # .1.5

Finally, unexclude the lines to see all of them:

```
***** ***** Top of Data *****  
000001 001  
000002 002  
000003 003  
000004 004  
000005 005  
***** ***** Bottom of Data *****
```

Enumeration patterns

The highlighted field on the first line defines a *pattern* used to describe how the enumeration process is to be done, in addition to providing an initial starting value. Enumerated values are assigned in ascending order by incrementing the starting value for each new line where subsequent enumeration values are stored. The enumeration pattern is in the form ***prefix digits suffix***, where the prefix and suffix are optional.

prefix The optional *prefix* may consist of any string of characters (including digits characters themselves) as long as the last character is a non-digit. Neither [\(EnumHexUc\)](#) nor [\(EnumHexLc\)](#) will adjust the casing of the prefix string; the value is left as-is. (For example, if you had a prefix of **0x** in a C program, [\(EnumHexUc\)](#) will not change **0x** into **0X**.)

digits For [\(Enum\)](#), *digits* are the characters **0-9**.

For [\(EnumHexUc\)](#) and [\(EnumHexLc\)](#), *digits* are the characters **0-9, A-F** and **a-f**.

At least one digit must be present. The *digits* value may contain a maximum of 9 full decimal digits or 8 hex digits, since the underlying arithmetic to increment values is done with 32-bit integers. The maximum decimal value is 4294967295 and the maximum hex value is FFFFFFFF. If the maximum value is reached and there are more lines to be enumerated, the number value wraps around to zero.

If sufficient data lines are enumerated to cause a maximum value to be developed (such as 999 in a 3-position decimal field) then the value will wrap around to zero and continue forward.

If a wrap-around-to-zero occurs, the zero and positive values will not have any leading zeros. If an exact size or number of leading zeros is required and wrap-around may occur, you may wish to perform the Enumeration in two steps; one from the initial value to the maximum value, followed by zero to the final value. Another approach is to allow the Enumeration function to operate over the entire range, and the overlay a line containing 0 characters in the required positions. The overlay command will only replace blanks with zeros, and so you can safely pad an Enumeration value to any number of desired leading digits.

The function [\(EnumHexUc\)](#) will adjust all *digits* greater than **9** to upper case **A-F**, even on the first (model) line. The function [\(EnumHexLc\)](#) will adjust all *digits* greater than **9** to lower case **a-f**, even on the first (model) line.

suffix The optional *suffix* may consist of any string of non-digit characters. Neither [\(EnumHexUc\)](#) nor [\(EnumHexLc\)](#) will adjust the casing of the suffix string; the value is left as-is.

Examples

- o **the Ada number 16#FF#**
the prefix is **16#**, the *digits* value is **FF** and the suffix value is **#**
- o **the C number 0xFF**
the prefix is **0x**, the *digits* value is **FF** and there is no suffix
- o **the Basic number &H123**
the prefix is **&H**, the *digits* value is **123** and there is no suffix
- o **the IBM assembler number X'FF'**
the prefix value is **X'**, the *digits* value is **FF** and the suffix value is **'**
- o **the simple string Page 1**
the prefix value is **Page..** (note that **.** represents blanks), the *digits* value is **1**, and there is no suffix

Handling of blanks in the prefix

The enumerate function will utilize blanks in the prefix to accommodate changes in length of the digits string as enumeration takes place. This is done to retain formatting as much as possible. Some examples of how this works (note that **.** represents *blanks*):

...&Hff	enumerates as	...&Hff	..&H100	..&H101	etc.
Page..8	enumerates as	Page..8	Page..9	Page.10	etc.
..(98)	enumerates as	..(98) ..(99)	.(100)		etc.
(..98)	enumerates as	(..98) (..99)	(.100)		etc.

Use of the ENUMWITH command

Normally, the [\(Enum\)](#), [\(EnumHexUc\)](#) and [\(EnumHexLc\)](#) functions increment each successive line by **1**. You can change the increment value to something other than **1** by using the [ENUMWITH](#) primary command. When you do this, the increment value applies to all edit sessions until you change it, but it will reset back to **1** when SPFLite is restarted. See [ENUMWITH - Change Increment for Enumerate Functions](#) for more information.

Application Note: Using a keyboard macro to automatically insert sequence numbers

You may have a file containing a fixed column of sequence numbers that need to be frequently renumbered. IBM ISPF has several "numbering" commands, such as **AUTONUM**, **NUMBER**, **NONUMBER**, **RENUM** and **UNNUMBER**. Because PC data is not generally "card-image based", like mainframe data, SPFLite does not support these commands. However, you can use the Enumerate feature of Power Typing to place (or remove) sequence numbers anywhere you wish.

You can automate the process of resequencing a file by placing the operation into a keyboard macro. Here is the general outline of how you could do this:

- o Define a keyboard macro to resequence your data; see example below
- o When you are likely to resequence the file several times, put labels like **.A** and **.B** to mark the beginning and end of the file, and enter Power Typing mode with **PTYPE .A .B**
- o If you resequence the file only occasionally, mark the block of lines using a **cc** block, and enter Power Typing mode with **PTYPE**
- o Press the key mapped to your keyboard macro

Suppose the data you want to resequence was a COBOL program that uses columns 1-6 for sequence numbers. Here is a macro that would enumerate columns 1 to 6. Let's assume you are going to map this to Ctrl-Shift N. So, you would enter KeyMap, and in the Ctrl-Shift N item, you would enter the following:

(TxtHome) [000001] (Left) (6:MarkLeft) (Enum) (Enter)

Here's what happens when you run this macro:

(TxtHome)

The cursor is placed in column 1 of the model line

[000001]

The initial value of the sequence gets stored in columns 1-6 of every line in the Power Typing range

(Left)

The cursor is moved from column 7 to column 6

(6:MarkLeft)

Columns 1-6 are highlighted, going in reverse order

(Enum)

All lines in the Power Typing range are numbered

(Enter)

This ends the Power Typing session

Using AUTOFAV to add to File Lists

When working on a project that involves many files, it can be convenient to have these files stored in a Favorites File List. However, if files are created frequently, it can be easy to forget to consistently issue a [FAVORITE](#) primary command every time. If that command is forgotten, your favorites list can be incomplete.

It is now possible to issue a [SET](#) command, so that whenever a file matching a particular file pattern (mask) is created or saved, a corresponding entry will be automatically created in a Favorites File List. Because SET names are a global resource, the AUTOFAV feature applies to every file that matches the file pattern(s), regardless of what profile they are associated with.

Setting up an AUTOFAV request

Creation of an AUTOFAV request is done with a **SET** command, using the format:

```
SET AUTOFAV.filenam = filelist-name
```

Here, **filenam** is a simple standard file mask string to specify what files are to be included, and **filelist-name** is the name of the File List to which the name is to be added.

For example, to place every new or saved file having a name like **ABC* .TXT** into the **ABC_PROJECT** File List, you would issue the command:

```
SET AUTOFAV.ABC*.TXT = ABC_PROJECT
```

You may create multiple SET entries with different masks, all referring to the same File List; this is not a problem.

NOTE: Because of the use of wildcard characters * and ?, it is possible that you may create multiple SET entries that could apply to a particular file-name. Which one takes precedence? The SET entries are maintained in alphabetical order, and **the first one which causes a match will be the one used**, even if a SET further down alphabetically is 'more specific'.

Example: Given the two SET entries,

```
SET autofav.*.txt = Temp1
SET autofav.ABC*.txt = Temp2
```

the file **ABC .TXT** would be added to the **Temp1.FLIST** even though the **ABC* .txt** mask is more specific.

Note: Because the SET list is maintained in alphabetical order, and gets re-sorted each time it is modified, the order you create your SET entries - whether by using the **SET** command or by direct editing in a SET Edit session - may **not** be the order in which the entries are stored. If you have any question on this, issue a **SET** command with no operands to enter a SET Edit session, **after** any changes have been made to it. That way, you will be able to see the (newly sorted) order of the SET names.

Using AUTOFAV with Named and standard File Lists

- When you wish to have a file recorded in a **named** File List of your choosing, simply use the File List name you have set aside for this. If the named File List does not yet exist, SPFLite will automatically create it the first time a file is saved when an appropriate **AUTOFAV** definition is in effect, unless you take some action to create it some other way.
- If you wish to have an **AUTOFAV** definition reference the **standard** ("unnamed") Favorite Files list, you can

spell out the full name of this list, as **Favorite Files**, such as, **SET AUTOFAV.XYZ*.TXT = "Favorite Files"**. If you do that, you have to include the name in quotes if you enter it on the command line. However, SPFLite understands the common abbreviations of this list, and so you can define this as any of the following, and they all mean the same thing as "**Favorite Files**":

- **SET AUTOFAV.XYZ*.TXT = Fav**
- **SET AUTOFAV.XYZ*.TXT = Favorite**
- **SET AUTOFAV.XYZ*.TXT = Favourite**

- The other predefined lists, such as **Open Files**, **Recent Files**, etc. cannot be used with **AUTOFAV**.
- **Named Favorites** cannot literally be used with **AUTOFAV** either, but that is not really a File List anyway, but rather, a "list of lists".

Working with the **SUBMIT** Command

Contents of Article

[Introduction](#)
[Configuring **SUBMIT** in the Global Options](#)
[The **SUBMIT** process](#)
[Including other files in the **SUBMIT** jobstream](#)
[Customizing **SUBMIT** for different File Types](#)
[Managing temporary files](#)
[Debugging the **SUBMIT** process](#)
[The **SPFSUBMIT** utility program](#)
[SPFSUBMIT syntax](#)
[Complementary Features for Hercules Users](#)

Introduction

Mainframe ISPF users are familiar with using the **SUBMIT** command to send JCL and data to be executed as a job under DOS/VS, MVS, VM or z/OS. The SPFLite primary command [SUBMIT](#) provides the ability to submit an edit file to an external process (which may be an executable program or batch file).

The [SUBMIT](#) facility is flexible enough to be usable in a wide variety of applications. Because of this, the notion of a "job" is very flexible, and may be configured as desired using SPFLite's Options GUI. The [SUBMIT](#) command may also take optional user parameters that could be used in a submit script if desired. A "job" **can** be, but need not be, a JCL job stream destined for a mainframe system. Alternatives to using **SUBMIT** might be [RUN](#) or [CMD](#) depending on your needs.

It is expected that a principle use of the [SUBMIT](#) command will be to submit jobs for execution on mainframe systems operating under control of the Hercules emulator program. More details on this are provided below. See [The **SPFSUBMIT** utility program](#). **SUBMIT** can be issued from an Edit, Browse or Clipboard session.

Configuring **SUBMIT** in the Global Options

Prior to using **SUBMIT**, you will need to determine just **how** you are going to be using it. If you're a Hercules user, skip to [The **SPFSUBMIT** Utility program](#) for details. Otherwise, you will probably be doing a simple invocation of a program or batch script file. So first, figure out exactly what the command should be to perform your desired function. Test the syntax using a normal DOS command window till you have it correct.

Then, set up this command in the SPFLite Submit options. (Enter Options => Submit) and you should see something similar to the following.

General | FM | **Submit** | Screen | KBD | Status | Schemes | HiLites |

Prototype command line to be used by SUBMIT

echo "~i"

Working Directory to be used by SUBMIT

D:\Documents\SPFLite\Jobs

CMD.EXE flags used by SPFLite CMD command

/K

CMD.EXE flags used by SPFLite RUN command

/K

Trigger key for the SUBMIT Include statement

#INCLUDE

Column number where above Include Key must appear

1



INI File is: D:\Documents\SPFLite\SPFLite.INI

Cancel

Done

Enter your tested command in the Prototype box. Replace the operand containing the filename to process with ~i. SPFLite will replace ~i with the temporary filename containing your edit text and invoke the command. The example below shows the prototype setup to use the SPFSUBMIT utility for Hercules users, use whatever command you tested above in the command window.

There are many other codes available in addition to ~i. All submit codes are optional, but it would be unusual to omit the ~i code. See the description of the [SUBMIT](#) command for a complete list of submit codes.

If you plan on using the SUBMIT INCLUDE feature, complete the two values to specify the INCLUDE trigger key. See [Options => SUBMIT](#) for further details.

The SUBMIT process

Because SPFLite does not know what the user's intent is in "submitting" a "job" it is necessary for you to define this meaning. The sequence of events that occur during a SUBMIT operation is as follows:

1. A file is opened for editing or browsing, or the contents of the Clipboard are opened for editing.

2. The desired line range is determined. This may be the entire file, or a subset of the file based on a line-label range, a **CC** block or **MM** block, or a set of lines defined by an ordinary or relative tag name. The line range may be further limited by excluded or non-excluded lines. The selected group of line is copied to a temporary file having the general form **SUBMIT_JOB12345.xxx**. (xxx will be the same filetype as that being edited) This file is written in the SUBMIT Working Directory as defined in the SPFLite Submit Options tab. This file is called the "submit input file".
3. A name is created for any messages that the external submit process generates in response to the submit action. If the **~R** or **^R submit-codes** are coded, SPFLite will create an empty file for it. This file is called the "response file". Names created as response-file names have the general form **SUBRESULTS_JOB12345.TXT**.
4. A 'prototype' command is generated from the information given in the OPTIONS-SUBMIT dialog described above (in the configuration section) This command may be a script command file like **.BAT**, or it may be an executable program.
5. Once the prototype command has been tailored, the command is executed. On the upper-right corner of the edit screen, you will see an initial message like, "**40 lines submitted for (JOB12345)**".
6. If your process utilizes a response file and writes to it during the submit process, SPFLite will detect the file being changed, and will display the first few lines produced by the submit process. e.g If you use **SPFSUBMIT** as your submit process, and redirect its *stdout* messages to the response file, you will see a popup message like "**SPFSUBMIT: 'ABC.JCL' submitted, 40 lines**" if the job submission was successful. Press OK on the popup message to proceed.

Including other files in the SUBMIT jobstream

SUBMIT allows you to embed / include external files into the jobstream during SUBMIT. To do this, you must define a unique identifier which will be used to trigger INCLUDE processing. This is done on the [OPTIONS=>SUBMIT](#) setting panel.

The default values for this trigger is the string **#INCLUDE** located in Column 1. You may of course set these to whatever you desire. The operand for **#INCLUDE** is simply the name of the file to be included. This value may be unquoted or quoted (single or double). If a simple filename, it must be located in the same folder as the file being edited. It can also be a fully qualified path / filename. Examples:

```
#INCLUDE MYFILE.TXT
#INCLUDE "C:\Users\Documents\Files\AnotherFile.txt".
```

Customizing SUBMIT for different File Types

You may be using SUBMIT to run local batch files against the text you are editing. An example might be to run the C compiler while editing a C source file. Setting up SUBMIT for this is straightforward, just specify the BAT file for the C compile in the SUBMIT prototype command.

But once you are using multiple languages, you certainly do not want your ASM files passed to the C compiler. What to do?

One of the settings available for a file type Profile is SUBARG, which can be any string you desire. Say you have several BAT files for different languages, such as ASMCOMP for ASM files, CCOMP for C files and BASCOMP for BAS files.

Set the SUBARG value for each of the file types to the appropriate BAT file name. Now you can specify the Prototype as (example):

```
C:\Users\Me\BatFiles\~Z.BAT ~i (The ~Z variable is replaced by
the SUBARG value.)
```

Now when you issue SUBMIT while editing an ASM file, it will use the ASMCOMP.BAT file. Similarly it will use the BASCOMP.BAT file for BAS files, etc.

Managing temporary files

Every time SPFLite is started up, it looks in the SUBMIT working directory for any temporary submit files more than 2 days old, and if found, deletes them. This means that you generally will not need to be concerned about these temporary files accumulating. If you wish, you can always manually delete these temporary files more frequently.

If your SUBMIT functions require supporting BAT or script files, this temporary directory is a convenient and relevant place to store them.

Debugging the SUBMIT process

If you use the **DEBUG** option, it causes the SUBMIT prototype command (such as **SPFSUBMIT.EXE**) to be run in a command window that stays open until you explicitly close it with a Windows **EXIT** command or by closing the command window by clicking on the X. This allows you to view any generated messages, run additional command-line programs, etc. if the external process launched by **SUBMIT** is not working properly.

The SPFSUBMIT utility program

Included in the distribution of SPFLite is the batch utility program **SPFSUBMIT.EXE** for submitting jobs to the Hercules mainframe emulator. This program has parameters and performs a function comparable to a utility supplied with Hercules, with the following differences:

- **SPFSUBMIT.EXE** uses SPFSUBMIT as the name of the environment variable with the host/port address
- Run-time messages and help are worded differently and include SPFSUBMIT in message text
- **SPFSUBMIT** assumes a default timeout interval of 5 seconds
- The return code values are zero for success and positive values when errors are detected
- Non-zero return codes are included in error message displays (such as, "RC=3")
- All run-time messages are written to *stdout* so that they can be redirected to an output file, such as the Response File represented by the SPFLite submit code ~R or ^R

When the SPFLite primary command **SUBMIT** is being used for simple job submissions to a single emulated mainframe system, it may be sufficient to call SPFSUBMIT.EXE directly as the prototype command that is defined in the [Options - Submit](#) tab of the Global Options dialog. When the submit requirements are more complex, the submit prototype command can launch a batch script that would in turn call **SPFSUBMIT.EXE**.

For **SPFSUBMIT** to successfully submit a job to an emulated mainframe system, the Hercules configuration file must have an emulated card reader device configured to read from a *socket* rather than from a PC disk file. A sample card reader device line in the Hercules configuration file might look like this:

```
000C      3505      localhost:3505  SOCKDEV  ASCII  AUTOPAD  TRUNC  EOF  # card reader
```

In the **SPFSUBMIT.EXE** command line, the "localhost" name can be specified literally as **localhost** or as **127.0.0.1**.

SPFSUBMIT.EXE can only submit a job when the target system is currently running. (There is no "job queue" in case the target system is not active.)

When **SPFSUBMIT** is executed with no parameters, a Help screen is displayed as follows, which explains how to use the program:

SPFSUBMIT syntax

The **SPFSUBMIT** program is used to submit a file to a given host:port address.

Command syntax:

```
SPFSUBMIT [-nnn] [host:port] file [file ...]
```

```
-nnn      Timeout value (1-999) in seconds; default is 5 seconds
host:port Target address to submit to; if not specified, the value
            of SPFSUBMIT environment variable is used, if defined
file      File being submitted
```

Examples:

```
SPFSUBMIT localhost:3505 job111.txt job222.txt
set SPFSUBMIT=localhost:2501
SPFSUBMIT job333.txt job444.txt
SPFSUBMIT myhost:3505 job555.txt
SPFSUBMIT 192.168.0.1:3505 job6.jcl job7.jcl
SPFSUBMIT -2 127.0.0.1:3505 C:\mypath\myjob.jcl
```

Return code values:

```
0  Successfully submitted
1  Cannot connect to host, bad socket address, or connection refused
2  Timeout value exceeded while trying to connect
3  Transmission error, or connection prematurely closed
4  File not found or cannot be opened
5  Submit failed, missing or invalid arguments, or unexpected failure
```

SPFSUBMIT messages are written on stdout and can be redirected
SPFSUBMIT with no parameters displays this help

Complementary Features for Hercules Users

The following features of SPFLite will prove useful to users of the [SUBMIT](#) command:

- The profile option **EOL AUTO** or **AUTONL** can be used to read SYSOUT files having nonstandard or inconsistent line terminations that include CR, LF and FF in unusual combinations. Hercules is known to create such files, and the **EOL AUTO** or **AUTONL** option works well for viewing them. When **EOL AUTO** or **AUTONL** is in effect, the first line of the file, and any lines read in that contained a Form Feed character will be marked with =PAGE> in the sequence area. You can use **UP PAGE** and **DOWN PAGE** to scroll up and down to the previous or next PAGE boundary. See [EOL – Set End-of-Line Handling](#) for more information. (The distinction between **AUTO** and **AUTONL** is in how lines terminated by a single CR are handled, as to whether this is a simple line end or if it implies overprinting.)
- The profile option **START NEW** can be used to automatically set a line label of **.START** on the last line of the file when the file is closed, and when reopened, SPFLite will automatically position the file to the **.START** label. This technique allows print spool files to be easily browsed at the point where new print information has been added by an outside process such as Hercules. See the [START](#) command for more information.
- The **SPFSUBMIT.EXE** program is provided with SPFLite to facilitate the submission of jobs for SPFLite to Hercules. The SPFLite installation will ensure this program is available to execute without requiring a full path, or you may specify a full path for this program if it is located elsewhere.
- SUBMIT-related commands [SUBCMD](#) and [XSUBMIT](#) were added to recent versions..

Writing a MACRO for a macro controlled string CHANGE

Structure of a MACRO for use in a Macro Controlled CHANGE

A macro to be used in a Macro Controlled CHANGE command (let's call these E-macros for short) communicates with the CHANGE command via two reserved macro functions:

Get_E_Source\$()

This macro function will return the value of the string which was successfully found by the CHANGE command

Set_E_Result()

This macro function is used to pass back to CHANGE the string which is to replace the found string.

A macro can utilize whatever logic is needed to determine what it will pass back as a string to be used by CHANGE.

Initialization and Execution Calls

Since the macro may have optional operands which may require validation, or have other activities which may need to be performed on a one time basis, the E-macro will be called once per CHANGE command to allow this initialization to be performed. Subsequently, the macro will be called once per found string to perform the actual CHANGE processing. These are called the Initialization and Execution calls.

For example, a command **CHANGE ALL P"@@##" E"MyMacro"** would be called once for Initialization and repeatedly for Execution as the CHANGE command locates each matching search string.

So how does the macro know which call is being made?

Since SPFLite FIND and CHANGE cannot 'find' a null string, the Initialization call is identified by **Get_E_Source\$()** returning a null string. Any returned non-null string is an Execution call.

Failing an Initialization Call

If, during the Initialization call, the E-macro determines it has a problem with the macro operands, or some other failure in initialization, it should exit the macro via a **Halt(FAIL, "Error message describing the failure")** command.

Successful Initialization Call

If Initialization is successful, the E-macro should exit with a **Halt("OK")** command.

Not all E-macros will require any processing at Initialization time, but must still be prepared to handle the call itself. All that is required for that is a simple one line at the start of the macro containing

```
if Get_E_Source$() = "" then Halt("OK")
```

Results from Execution Calls

There is no support for error returns from Execution calls. Your E-macro must return a new CHANGE value via **Set_E_Result()**, even if it is simply a copy of the found string, or null. It may return an Error message, but doing so will not terminate CHANGE processing. i.e. a CHANGE ALL will continue normally.

Example / Case study

Following is an example of an E-macro. This macro performs a table lookup to locate and return a replacement string for the found string. The table of old/new string values is maintained in an external table file so that the E-macro itself does not need to be altered as the table data changes.

Since the table is an external file, we certainly do not want to have to Open and read the file for every CHANGE command, so the macro will only do this on the Initialization call and will keep the table resident. This means all Execution calls will not require external file access.

The Initialization code starts off with:

```
str = Get_E_Source$()                                ' Get the passed found
string
if str = "" then                                     ' If null, its the
initialize call
    TableLoaded = Get_Gbl_Num("TableLkupOK")          ' See if table already
loaded
    if TableLoaded > 0 then                           ' Yes, say all is well
        Set_E_Result("OK")                            ' and
        Halt(OK)                                     ' exit quickly
    end if
```

It starts with the standard test for the Initialization call and then checks the Global variable *TableLkupOK* which is used to track whether the table file has already been loaded. If it has, it simply returns "OK" to indicate initialization is complete.

Next is the routine to load the external file when needed

```
'----- Need to load the table file
fHandle = FILE_OPEN("D:\Documents\SPFLite\Macros\TableLkup.Data", "INPUT")
if fHandle = 0 then halt(fail, "TableLkup.Data load failed, cannot open file")
do while isfalse file_eof(fHandle)                  ' read file
    dline = FILE_LineInput(fHandle)                 ' Get a line

    '----- Create a lookup key by preceding the string with "TL_"
    operand1 = "TL_" + parse$(dline, ", ", 1)        ' Build key by prefixing
with TL_
    operand2 = parse$(dline, ", ", 2)                 ' Extract the return data

    Set_Gbl_Str(operand1, operand2)                  ' Save entry in table in
Global storage
loop
i = FILE_Close(fHandle)                           ' Close file, we're done
Set_Gbl_Num("TableLkupOK", 1)                     ' Remember we've loaded the
table
Set_E_Result("OK")                                ' Tell mainline we're done
Halt(OK)
```

This code uses the standard thinBasic FILE module for File access. If the file does not open successfully, it shows how to exit with an error message.

The table is stored using the normal SPFLite Global string support. The lookup keys are prefixed with TL_ to make them unique in global storage.

```
' The external file is only loaded once per SPFLite session. The format is
simply
' 2 entries per line separated by a comma. e.g.
'  AA12,ABCD
'  AA13,ABCE
```

```
' BB01,DEFG
' CC99,HIJK
' XX00,ZZZZ
```

The remainder of the macro is the execution call

```
'----- It's the normal execution call
else
    operand1 = "TL_" + str
    operand2 = Get_Gbl_Str$(operand1)
    ErrMsg = Get_Msg$
    if Get_RC = 0 then
        Set_E_Result(operand2)
        halt(ok)
    else
        Set_E_Result(str)
    end if
string
    halt(Fail, "TableLkup: " + ErrMsg + " in line: " +
format$(Get_LNum(Get_Find_Lptr)))  ' Else error
    end if
end if
```

This routine simple uses the found string to build the table lookup key, fetches the replacement string and passes it back via Set_E_Result().

If the lookup fails, it passes back the original string and issues an error message. Even though an error does not interrupt a CHANGE ALL type command, the error messages are stacked and made available to the user via a HELP command immediately after the CHANGE command completes.

Here's the complete E-macro for review:

```
' TableLkup.macro
'
' This macro shows how to use an E'xxxx' macro to provide a table lookup
' facility. The macro is invoked as part of a normal CHANGE command
' e.g. CHANGE P'@#@##' WORD ALL E'TableLkup'
' This command searches for all 'words' consisting of 2 Alpha and 2 numeric
characters,
'
' The located words are passed to this routine to be replaced via a table lookup
of
' an external table (in this case in the file 'TableLkup.data')
' The external file is only loaded once per SPFLite session. The format is
simply
' 2 entries per line separated by a comma. e.g.
' AA12,ABCD
' AA13,ABCE
' BB01,DEFG
' CC99,HIJK
' XX00,ZZZZ

uses "FILE"                                ' Attach the FILE module
dim str, dline, operand1, operand2, ErrMsg as string
dim fHandle as dword
dim i, TableLoaded as number

'----- See if this is an initialize call, if so, load the table
str = Get_E_Source$()                         ' Get the passed found
```

```

string

if str = "" then                                ' If null, its the
initialize call
    TableLoaded = Get_Gbl_Num("TableLkupOK")      ' See if table already
loaded
    if TableLoaded > 0 then                      ' Yes, say all is well
        Set_E_Result("OK")                         ' and
        Halt(OK)                                 ' exit quickly
    end if
    ----- Need to load the table file
    fHandle = FILE_OPEN("D:\Documents\SPFLite\Macros\TableLkup.Data", "INPUT")
    if fHandle = 0 then halt(fail, "TableLkup.Data load failed, cannot open file")
    do while isfalse file_eof(fHandle)            ' read file
        dline = FILE_LineInput(fHandle)            ' Get a line

        ----- Create a lookup key by preceding the string with "TL_"
        operand1 = "TL_" + parse$(dline, "", 1)      ' Build key by prefixing
with TL_
        operand2 = parse$(dline, "", 2)              ' Extract the return data

        Set_Gbl_Str(operand1, operand2)              ' Save entry in table in
Global storage
        loop
        i = FILE_Close(fHandle)
        Set_Gbl_Num("TableLkupOK", 1)                ' Close file, we're done
        ' Remember we've loaded the
table
        Set_E_Result("OK")                          ' Tell mainline we're done
        Halt(OK)

        ----- It's the normal execution call
else
    operand1 = "TL_" + str
    operand2 = Get_Gbl_Str$(operand1)
    ErrMsg = Get_Msg$
    if Get_RC = 0 then
        Set_E_Result(operand2)
        halt(ok)
    else
        Set_E_Result(str)                         ' Pass back unchanged
string
    halt(Fail, "TableLkup: " + ErrMsg + " in line: " +
format$(Get_LNum(Get_Find_Lptr)))                  ' Else error
    end if
end if

```

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Print Screen Functions

SPFLite provides several types of Print Screen capabilities. These are implemented as primitive keyboard functions, which means you can assign each function to any desired key combinations.

These functions are:

(PrtScrnClipboard)	Will send a text-format image of the entire edit screen to the Clipboard.
(PrtScrnLog)	Will send (append) a text-format image of the entire edit screen to the log file (SPFLiteScrPrt.LOG) in the SPFLite data directory.
(PrtScrnPrinter)	Will send a text-format image of the entire edit screen to your default SPFLite printer.
(PrtTextClipboard)	Will send only the actual data from the currently visible text lines to the Clipboard.

For information on how to assign these functions to your choice of keys, see "[Keyboard Customization](#)".

Keyboard functions that involve the clipboard, such as [\(PrtScrnClipboard\)](#) and [\(PrtTextClipboard\)](#), can accept a Named Private Clipboard operand, such as **(PrtScrnClipboard/myclip)** and **(PrtTextClipboard/myclip)**. See [Windows Clipboard](#), [Cut and Paste](#) and [List of Keyboard Primitives](#) for more information.

Using the standard Windows Print Screen functionality

If you want to do a standard Windows Print Screen operation, then before pressing the Print Screen key, you should move the Windows focus **away** from the SPFLite window. Doing so prevents SPFLite from "capturing" the Print Screen key-code and acting on it, allowing Windows to respond to it normally.

If you want to do a standard Windows Print Screen of any part of SPFLite itself, the physical Print Screen key must be mapped to **(Null)**. This applies to any "chords" of Print Screen as well - if you want Windows to see the **Alt Print-Screen**, then the Alt Print-Screen entry in the SPFLite KeyMap must be set to [\(Null\)](#).

Note: We use this technique ourselves to capture screen shots of SPFLite to include in this Help document. Otherwise, SPFLite would intercept the key, Windows would never see it, and the Print Screen functionality of Windows would not occur.

If you need to do a standard Windows Print Screen of any part of SPFLite itself, but **also** want to have any of the Print-Screen function available as described above, you **can** do this, provided the physical Print Screen key is mapped to [\(Null\)](#), and then any of the SPFLite Print Screen keyboard functions must be mapped to any key(s) **other than** the physical Print Screen key.

If you are looking for an alternative key to map the print-screen functions to besides the physical Print Screen key, you might want to consider the **Pause** key. This key is fully mappable by SPFLite, virtually no other software makes use of it, and SPFLite itself does not assign any defaults to it.

Creating and Replacing Data Files

Use the [CREATE](#), [REPLACE](#) or [SAVEAS](#) primary commands to specify a file to be written from the data being edited. [CREATE](#) writes a new file while [REPLACE](#) rewrites a file if it exists, or creates it if it does not already exist. The [SAVEAS](#) command will create a new file and immediately switch the current Edit Tab to the newly created file. The process of creating and replacing data is very similar. However, remember that when you replace data, the original data is deleted and replaced with the new data.

[SAVEAS](#) always writes the entire contents of the edit file, but the [CREATE](#) and [REPLACE](#) commands can use all the selection capabilities of the line-range-operands to specify what lines are to be written.

When a **SAVE** command is issued for a newly created file that does not yet have a name (you will know this by the file-tab label of **(Empty)** for this file), you will see the same file-saving dialog that you will see when **SAVEAS** is used. A window will appear with a title of **Specify file to SaveAs**, where you can enter the new file's name.

Enter [CREATE](#), [REPLACE](#) or [SAVEAS](#) on the Command line, followed by the name of the file to be created or replaced. For the **CREATE** and **REPLACE** commands you can also use all the selection capabilities of the line-range-operands to specify what lines are to be written. If you omit the line numbers, you can use the [C/CC](#) or [M/MM](#) line commands to specify which lines are to be copied or moved. Then press Enter.

If you omit the file name with the [CREATE](#), [REPLACE](#) or [SAVEAS](#) commands, the editor displays a standard Windows Save As dialog requesting the file name you want created or replaced.

When no operand at all or only a simple unqualified filename is provided for the **CREATE** command, the default directory used for writing the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the path for *that* active file is used as the default for the command.
- If there is **no** active file (i.e. the tab header displays **(Empty)**), the current displayed directory of File Manager will be used.

In addition to these commands, you can save every open edit session by using the **SAVEALL** command. See [SAVEALL - Save All Current Tabs](#) for more information.

Including (Copying) Another File

While you are editing, you can copy another file into the current data by using the [COPY](#) primary command. The location at which it is to be inserted is specified with the [A](#), [B](#), [AA](#), [BB](#), or [H/HH](#) line commands. See those line commands for details of how each interact with the **COPY** command.

If the [COPY](#) command is issued when there are **no** lines in the edit work area, then no destination line commands are needed.

Refer to the [COPY](#) primary command for further details.

Word Processing Support

Contents of Article

[Introduction](#)
[Case sensitivity and default search context](#)
[Mouse-based text editing](#)
[Formatting paragraphs](#)
[Splitting lines](#)
[Joining and gluing lines](#)
[The GLUEWITH command](#)
[Case-conformant change strings](#)
[Performing line swaps](#)
[Performing text swaps](#)
[Text-swap example](#)
[Text Move example](#)
[Using raw-mode copying](#)
[Application Note: Splitting Lines Based on a Search String](#)

Introduction

SPFLite is not designed as a word-processor per se, since its primary focus is on editing **lines** rather than **words**, just as in IBM ISPF. However, a number of word-processing features have been added, so that when you need to do simple word-processing tasks, you will have less need to leave SPFLite for some other editor (like Notepad) and then come back.

This section is a general overview of the commands used for word or text processing. These commands consist of the line commands: **TF** (text flow), **TS** (text split), **UC** (Upper Case), **LC** (Lower Case) **SC** (Sentence Case) and **TC** (Title Case). As Primary commands there are the companion commands **UC**, **LC**, **SC**, and **TC**. And as keyboard primitives we have (**UpperCase**), (**LowerCase**), (**SentenceCase**) and (**TitleCase**). **TF** now also comes in a block-mode form, the **TFF** command and new commands **Text Break** (**TB**) and **Text Margin** (**TM**) are also available.

You can take advantage of implicit highlighting of single characters. When character-modification functions are used when the cursor is in the data area, but no data is actively being highlighted, the single character at the cursor position is modified as if that character were highlighted. Making such small changes is now easier, faster and more reliable. Functions affected by this are any that modify data that is dependent on a selected field, such as word-processing functions (**UpperCase**) and (**LowerCase**), and color modification functions like (**PenRed**).

The 'casing' commands allow quick changing of the case of a portion of text into alternative formats. **SC** reformats in **sentence** format: the first letter of the first word of a sentence is capitalized, and all other characters are lowercased. **TC** reformats as a Title: the first letter of **each** word is capitalized. **UC** and **LC** are conventional Upper Case and Lower Case commands.

TF and **TS** assume that the data is grouped in paragraphs. A paragraph is a group of lines that begin in the same column. The first line of a paragraph is excluded from the grouping. The editor interprets any indentation or blank line as representing a new paragraph. It also recognizes word processor control words used by various common scripting languages. These control words begin with a period, a colon, a '<', or an ampersand. Also, additional "paragraph-based" line commands **Text Break TB/TBB** and **Text Margin TM/TMM** are available.

If you use text line commands frequently, you can assign commands like **TS**, **TF**, **TB** and **TM** to function keys. (Assigning block-mode commands like **TFF**, **TBB** and **TMM** is possible but not as useful.) Use **KeyMap** to open the keyboard preferences dialog.

See also [Using TM to Emulate the TE command on ISPF](#) for notes about simulating the ISPF Text Entry line command.

See also [Application Note: Splitting Lines Based on a Search String](#).

Case sensitivity and default search context

You can use the **CASE** command so that unquoted strings, like **ABC**, or quoted strings without a type code, like '**ABC**' will be assumed to be case-sensitive, as if specified as **C 'ABC'**, or case-insensitive as if specified as **T 'ABC'**. These defaults are set with **CASE C** or **CASE T**, respectively. See the [CASE](#) command for more information.

FIND and **CHANGE** commands take an option that describes the [search context](#), which is the conditions under which SPFLite decides that a string of characters is the one you want. The deciding factor is whether a string is delimited or not, and if so, where. A delimiter is either (a) any character **not** listed on the **WORD** line, or (b) a blank, or (c) the left or right edge of the line. The choices you get basically allow you to either require a delimiter or they require the absence of a delimiter. With that in mind, the four standard search contexts are:

Search Context Operand	Delimiter present on left?	Delimiter present on right?
CHARS	Don't care	Don't Care
WORD	Yes	Yes
PREFIX	Yes	No
SUFFIX	No	Yes

By default, SPFLite normally assumes the **CHARS** context; that is, it doesn't care what is on the left and right side of string. If the string is there, it's found. If you are working extensively with words, it would be convenient if you didn't have to keep saying **FIND ABC WORD** all the time.

To let SPFLite know that you want to assume you are working with words unless you say otherwise, you can say **FIND WORDS** or **FIND CHARs** to temporarily change this assumption. See the [FIND](#) command for more information. To permanently change this assumption, you can set the [Use WORD as the default for FIND/CHANGE commands](#) checkbox in the [Options - General](#) tab of SPFLite Global Options.

See the [FIND](#) command and the [WORD](#) command for more information.

Mouse-based text editing

The mouse can be used to quickly perform certain editing tasks faster than using the keyboard alone. You are invited to try these techniques yourself to see how useful they can be:

- If you highlight part of a line with the mouse, you can delete it using the **DEL** key, or replace it with any single data character by typing it. So, if you had a string like **aaaXXXbbb** where the XXX part was highlighted, typing the **DEL** key would produce **aaabb** and typing a / slash character would result in **aaa/bbb**.
- If you double-click on a 'word' string (characters that are bounded by **non-WORD** characters or by the edge of the line) the whole word will be highlighted.
- If you highlight a 2-dimensional block of characters, the **DEL** key will delete the whole square block. If you type any single data character, that character will appear once on each line of the block, the same as if the block were first entirely deleted and then replaced by a vertical column one character wide, one for each line of the block. So, if you had a block where **aaaXXXbbb** appeared on each line in the block and the XXX part was highlighted, typing the **DEL** key would produce **aaabb** on each line of the block, and typing a / slash character would result in **aaa/bbb** on each line of the block.

- If you highlight a 2-dimensional square block of characters, you can copy the whole block into the clipboard. You can then paste that whole square block into another location, using the [\(Paste\)](#) function.
- If you highlight part of a line, or 2-dimensional square block of characters, you can replace the highlighted area with an equal number of blanks, by using a key mapped to the [\(Erase\)](#) function.
- If you highlight part of a line with the mouse, you can find additional instances of that string using the [\(FindNext\)](#) and [\(FindPrev\)](#) functions.
- You can use the mouse to quickly enter frequently-used line commands. For example, the line commands **CC** and **MM** are often used on blocks of lines. By setting selected mouse-button actions, a single mouse click can set a command on a line. For example, suppose that Ctrl+Left Mouse Button were mapped to **{CC}**. Then, by just holding down the Ctrl key, you can quickly set the boundary of a **CC** block by clicking the left mouse button on the desired line.

Formatting paragraphs

The **TF** (text flow) line command formats paragraphs. It assumes that the sentences are roughly in paragraph form with a ragged right margin when it attempts to recognize groupings. **TF** can be followed by a number (**TF72** for example) to specify the desired right side column for the paragraph. If you do not specify a number, the current right side of the edit screen is used.

The editor assumes that because the first line of a paragraph may be at a different indentation level than the remainder of the paragraph, the starting column of the second line is the left side of the paragraph. When formatting paragraphs, the editor:

- Moves text so that each line contains the maximum number of words. **TF** limits its activity to within the limit described above. Thus, it can be used to flow text within a border.
- Keeps any blanks between words.
- Assumes one blank between the word at the end of a line and the word on the next line, except when the line ends with a period. In that case, the editor inserts two blanks.

The end of the paragraph is denoted by a blank line, a change in indentation, or the special characters period (.), colon (:), ampersand (&), or left carat (<) in the left boundary column. These special characters are commonly used as control indicators by a variety of scripting languages and formatting utilities.

The restructure operation removes trailing blanks on a line by using words from the following line. It does not remove embedded blanks within a line. Accordingly, if one or more words in a line are to be removed, delete the words rather than type over them.

You can use the **TFF** block command to perform the same paragraph formatting done by **TF**, but applied to multiple paragraphs.

You can also use the Text Margin line command **TM/TMM** to format paragraphs much like **TF** does, but in a way that is less sensitive to any non-default **BOUNDS** margins that may be in effect. See [TM / TMM - Set Text Margin](#) for more information.

Splitting lines

The Text Split line command **TS** splits a line into two lines. The cursor shows where the line is to be split. The editor moves the characters to the right of the cursor or to a new line following the original line and aligns the new line with the left side of the paragraph. As mentioned earlier, the left side of a paragraph is determined by looking for a pattern in the lines preceding or succeeding a paragraph.

One or more blank lines are inserted after the line being split, depending on what you specify when you enter the **TS** command. If not used, these lines will be removed (as are all inserted temporary lines). You can also specify

a zero line count using **TS0** to split lines without inserting any blank lines between them.

Most users will find it faster and more convenient to map the **TS** command to a key (such as Alt-S, for example) and to move the cursor to the desired split point by using the mouse.

You can map mouse buttons with any combination of Shift, Ctrl and/or Alt modifiers. As an example, you could map the Alt+Left Mouse Button to `{TS} (2:Enter)`. By doing this, you can quickly split any line by holding down the Alt key, moving the mouse pointer to the desired location and pressing the Left Mouse Button.

The Text Break line command [TB/TBB](#) also breaks apart lines like **TS** does, but when blank lines are inserted at the break point, these are permanent blank lines rather than temporary ones. The **TBB** form can be used to break apart of block of lines all at the same column location. Like **TS**, **TB/TBB** will take an **n** value of **0** to insert zero blank lines.

You can also use the [SPLIT](#) primary command to split lines based on a Picture string.

Joining and gluing lines

The existing **Join** and **Glue** line commands **J/JJ** and **G/GG** perform "physical" join and glue operations. That means that for any two lines being physically combined, the data on the original lines themselves is not changed.

In addition to "physical" join and glue, you can perform a "text join" or "text glue" using **TJ/TJJ** and **TG/TGG**. When lines are combined in "text" mode, the "join point" of each line (the point at which the two lines are combined together) are trimmed of blanks. For example, if line 1 and line 2 are being joined or glued together in text mode, all trailing blanks are deleted from the right side of line 1, and all leading blanks are deleted from the left side of line 2.

With these meanings in mind, here is what the Join and Glue command do:

The [J/JJ](#) (physical join) line command concatenates lines together, with one intervening blank at the join point of each line. The contents of the original lines are not changed.

The [G/GG](#) (physical glue) line command concatenates lines together, with no intervening blanks at the join point of each line. The contents of the original lines are not changed.

The [TJ/TJJ - Text Join Lines](#) command concatenates lines together, with one intervening blank at the join point of each line. The original lines are trimmed of leading or trailing blanks (as discussed above) at the join point.

The [TG/TGG - Text Glue Lines](#) command concatenates lines together, with no intervening blanks at the join point of each line. The original lines are trimmed of leading or trailing blanks (as discussed above) at the join point.

You can also use the [JOIN](#) primary command to join lines based on a Picture string.

The **GLUEWITH** command

By default, the Glue line commands **G/GG** and **TG/TGG** concatenate glued lines together with no intervening blanks or other characters. The **GLUEWITH** command can be used to define a string to be inserted between such glued lines.

The **GLUEWITH** string is a global value, and has an installation default of "" (a zero-length string). This value is stored in the SPFLite.INI file, and is thus persistent. The only way to view or modify the **GLUEWITH** string is by using the **GLUEWITH** primary command in an edit session.

The **GLUEWITH** string setting applies only to the **Glue commands G/GG and TG/TGG**, not to the **Join commands J/JJ and TJ/TJJ**.

Case-conformant change strings

When you do a change with search string having a type code of T, it matches letters in a case-insensitive way. So, if you say,

```
CHANGE WORD T'four' 'nine'
```

it will match on the word "four" no matter how it is capitalized. However, regardless of the original string, the result of this CHANGE will always be capitalized as "nine":

```
four becomes nine
Four becomes nine
FOUR becomes nine
```

and so, the result fails to *conform* to the character-casing of the original string.

With *case-conformant changes*, you can make the result string match the pattern of upper and lower casing that existed in the original string. That is, NOW you can make *this* happen:

```
four becomes nine
Four becomes Nine
FOUR becomes NINE
```

How? Just put a type code of T on the change string, like this:

```
CHANGE WORD T'four' T'nine'
```

You can find more information in the [Case-Conformant Change Strings](#) section of [Finding and Changing Data](#).

Performing line swaps

SPFLite provides an easy way to swap data from two different areas. The types of data swaps that are supported are line swaps and text swaps. See [M / MM - Move Lines](#) and [W/WW - Swap Lines](#) for more information on line swaps.

Performing text swaps/moves

A *text swap* occurs when a single-line or multi-line selection of text is swapped with another another one. If a single-line text selection is involved, the data can be either on the same line or on another line. When both segments are on the same line, no overlap may occur.

Note: It is also possible to swap a marked block when no second block is highlighted, effectively making the swap activity into a text-move function. The marked block is moved to where the cursor is currently located, and text to the right of it is pushed over to make room.

Even with traditional editors like Notepad with its Windows-oriented cut-and-paste functions, it's not an easy thing to do, but SPFLite makes this task quite simple. The process works like this:

1. A keyboard primitive command ([Swap](#)) is now defined.
2. The ([Swap](#)) command must be mapped to a key. There is no default mapping for ([Swap](#)); you must map this yourself. For sake of discussion, let us assume this key is Alt-W, though any desired key may be used.
3. Highlight a piece of text on a single line or on multiple lines. Text selection can be done with the mouse, with shift-arrow keys, or with the line command [T/TT](#). Then press Alt-W.
4. The highlighting used for the text that is going to be swapped changes to underscores, and a message of

Swap pending appears on the status line.

5. Now, highlight a second piece of text on a single line or on multiple lines, and press Alt-W again. Now, the two sections of text are swapped, all highlighting coloring is removed, and the **Swap pending** status disappears.
6. **Note:** The two areas of text may, but need not, have the same width. As seen in the example below, the strings GROSS-PAY and NET-PAY are swapped, even though they are of different sizes. The second marked area may not even **be** an area. If no area is selected but the cursor remains within the normal text area (and not within the 1st marked area), then it is treated as a Null marked area and the cursor indicates **where** the 1st marked area is to be moved.
7. If you are swapping a 2D multi-line block of text, follow the same procedure as is described above. When you get to the point where you are defining the second two-dimensional block, SPFLite know how many lines are involved in the first two-dimensional block, and will automatically highlight that many lines in the second block.

For example, if you first define a block of 10 columns by 5 lines, when you begin to highlight the second block, SPFLite knows the first block is contained on 5 lines, so as you go to highlight the second block, it will automatically highlight 5 lines, starting on the line where you begin highlighting. This is done so you don't have to manually try to "get the size right" on the second block - this is done for you. The two blocks need not be the same width, just as equal widths are not required for single-line text swaps.

8. **Special considerations:** If you do the first part of the text-Swap, and the **Swap pending** status was visible, and **then** you decide to change your mind and not finish it, you can "back out of it" in one of two ways: (a) While ensuring that no 'normal' highlighting is in effect, type Alt-W again. Since there would be no second text block with which the first block is to be swapped, the second Alt-W in this case is considered as a "cancel" request; the **Swap pending** status disappears and all highlighting is removed. (b) Text Swap is canceled in response to an ordinary **RESET** command; no special keywords need to be added.
9. The Swap-pending status is not persistent and not saved in the STATE information.

This might sound a little involved, but once you understand the process, you will see that text swapping is actually very easy, convenient and fast. Try it!

Text-swap example

000001 MOVE **GROSS-PAY** TO **NET-PAY**.

Highlight first field via mouse:

000001 MOVE **GROSS-PAY** TO **NET-PAY**.

Invoke the [\(Swap\)](#) function (via Alt-W in the sample key mapping); this causes the highlighted field to be underlined instead:

000001 MOVE **GROSS-PAY** TO **NET-PAY**.

At this point, the status line contains: **Swap pending**

Now, highlight second field via mouse:

000001 MOVE **GROSS-PAY** TO **NET-PAY**.

Invoke [\(Swap\)](#) function (via Alt-W) a second time; fields are swapped; underscores and highlighting are now removed:

000001 MOVE NET-PAY TO GROSS-PAY.

At this point, the status line no longer contains: Swap pending

Text-move example

This is similar to the previous example but demonstrates using a null block for the 2nd marked area to cause a Move operation to take place.

000001 MOVE GROSS-PAY TO RIGHT-HERE.

Highlight first field via mouse:

000001 MOVE **GROSS-PAY** TO RIGHT-HERE.

Invoke the [\(Swap\)](#) function (via Alt-W in the sample key mapping); this causes the highlighted field to be underlined instead:

000001 MOVE GROSS-PAY TO RIGHT-HERE.

At this point, the status line contains: Swap pending

Now, move the cursor to the insertion point (the 1st character of RIGHT-HERE):

000001 MOVE GROSS-PAY TO RIGHT-HERE.

Invoke [\(Swap\)](#) function (via Alt-W) a second time; the 1st marked field is moved to the insertion point:

000001 MOVE TO GROSS-PAY RIGHT-HERE.

At this point, the status line no longer contains: Swap pending

Using raw-mode copying

You can copy data in what is termed Raw Mode. Raw mode copy means that there are no End of Line terminators inserted into the copied text. So, when you copy text that encompasses more than one line, all the text from all of the lines are effectively glued together in one long character string. This action happens under the following cases:

- **CUT** primary command with **RAW** keyword
- The [\(CopyRaw\)](#) keyboard function
- The [\(CopyPasteRaw\)](#) keyboard function

When you copy text in Raw Mode, you can then go outside of SPFLite and paste the text thus copied as if were a single line, for example, into a Notepad session. By doing this, you would be able to perform a type of conversion from the kind of individual lines that SPFLite edits to text that may wrap across multiple lines, as editors such as Notepad commonly deal with.

Application Note: Splitting Lines Based on a Search String

(Prior to introduction of SPLIT / JOIN commands)

Sometimes you may need to split one or more lines apart based on a string. For example, a common data format is called Comma Separated Values, or **CSV**. As the name implies, these are files with data fields separated by commas. Suppose you had a file of such values and wanted the fields to be split apart on separate lines. You could manually position the cursor each place a comma appears, and issue a Text Split line command **TS** mapped to some key. If you had to do this for many lines, it could be very time-consuming.

Since the release of [**SPLIT** / **JOIN**](#), these tasks are now facilitated by the **SPLIT** and **JOIN** edit primary commands.

See [Working with SPLIT and JOIN Commands](#) for more information.

Working with Virtual Highlighting Pens

Contents of Article

[Introduction](#)

[Colors and color palettes](#)

[Colors and LOCATE](#)

[Do not confuse virtual highlighting pens with automatic colorization](#)

[Avoid confusing yourself with clashing colors](#)

[Example of using highlighting pens](#)

[Resetting highlighting pen colors back to the default](#)

[Highlighting pens and special situations](#)

[NOTE lines and highlighting pens](#)

Introduction

In the "good old days" when people worked with big printouts, it was often necessary to mark them up with highlighting pens. SPFLite now allows a way to do this to files with "virtual" highlighting pens and with optional highlight requests with **FIND** and **CHANGE** commands.

You can mark (Hi-Lite) a section of text in any of 15 different colors. Prior versions of SPFLite only supported **RED**, **BLUE**, **GREEN** or **YELLOW** but current releases support additional colors.

The name **STD** is used to 'clear' a color hi-lite. Setting a hi-lite consists of marking the desired area and pressing a key to which a (Pen/colornamei) function has been assigned. e.g. (Pen/BLUE) (Pen/VIOLET). (Pen/STD).

Any of the defined color names can be added as keywords to any Primary command which supports searching (like **FIND**, **CHANGE** etc.)

Types of Color Operands

colorname

When a **colorname** (like **GREEN**) is added to a Primary command like **FIND**, it requests that the string only be considered "Found" if it is currently hi-lited **IN** that specified color.

-colorname

When a **-colorname** (like **-BLACK**) is added to a Primary command like **FIND**, it requests that the located string be in any color **other than** the specified color (including uncolored).

+colorname

When a **+colorname** (like **+BLUE**) is added to a Primary command like **FIND**, it requests that the located string (regardless of its current color) be hi-lited in the specified color.

Using **RED** as an example:

- **FIND ABC RED** means to look for a string that is entirely Red. Whether the string is found or not, or is entirely Red or not, its current color is not changed.
- **FIND ABC +RED** means to make the **ABC** find string Red, and **CHANGE ABC DEF +RED** will make the **DEF** change string Red. A string's color is only changed if a color name with a **+** plus sign is used.
- **FIND ABC -RED** means to look for a string that is **not** (entirely) Red. This includes string that might have a mixture of colors. Whether the string is found or not, its current color is not changed.

- **FIND ABC SOLID** and **FIND ABC -SOLID** refer to strings that are (or are not) entirely one color (as opposed to being of mixed colors) without limiting the search to any particular solid color. Whether a solid-color string is found or not, its color is not changed. Because **SOLID** is a "generic" description and not a specific color like **RED**, you can **FIND** strings that are, or are not, **SOLID**, but you can't **CHANGE** something to a "color" of **SOLID**, because it isn't any particular color.

Don't forget the **other** things you can do with your "virtual printout" file. Line labels and line tags are like "paper clips", and **NOTE** lines can be used to "scribble" on the "printout" like you would with a "sticky note". See [Working with Line Labels](#), [Working with Line Tags](#), and [Working with NOTE Lines](#) for more information.

The colors you set are persistent, may be found with **LOCATE** color options, and may be cleared with **RESET** color options. As with all persistent attributes of a file, the **STATE** profile option must be **ON** to enable persistent colors. If **STATE** is **OFF**, you can still put colors on your text, but when you close the file and reopen it, the color information will be discarded.

Colors and color palettes

The exact color palette used for any of the hi-lite colors can be tailored in the Options -> HiLites settings, so you can adjust the colors to maximize readability. When changing these values, the Options dialog will display sample text so you can readily see just what the chosen colors will look like.

Colors and LOCATE

Once your text is colored, you can use **LOCATE** to find lines having any text with a particular color. **LOCATE** allows the use of **STD** to indicate lines which have only standard coloring on them. You can also find lines which have any sort of coloring on them at all by saying **LOCATE NOT STD**.

You can use **LOCATE ALL** to exclude or unexclude lines based on their 'condition', which includes their color.

See [LOCATE - Scroll to a Specific Line](#) for more information.

Do not confuse virtual highlighting pens with automatic colorization

When you edit a file such as program source code, and you have an automatic colorization file defined for that file type, SPFLite renders your data lines according to the syntax and color choices defined in the **.AUTO** file.

Suppose one of the lines in your **.AUTO** file defined the keyword **FUNCTION** to be displayed in red, like this:

WORD 4 FUNCTION

*-- where 4 means "red" in an **.AUTO** file*

Now, suppose you have a data line with the keyword **FUNCTION** on it and it is being displayed in red.

Then you say **LOCATE RED** to find this line. Will it work? **No**. One is a function of colorization, the other from specific Hi-Lite actions.

Avoid confusing yourself with clashing colors

The exact colors displayed by the highlighting pens can be adjusted in the Options -> HiLites dialog to anything you wish. However, if you choose the same colors as SPFLite uses for text selection, for indication of found text during a **FIND** or **CHANGE** operation, or the colors displayed as a result of an active Automatic Colorization file (a **filetype.AUTO** file), it could get very confusing.

It will probably be best to avoid those particular colors. If you really need to apply highlighting pens to a file that has **HILITE AUTO** enabled, you may wish to temporarily issue a **HILITE AUTO OFF** so that you don't run into a

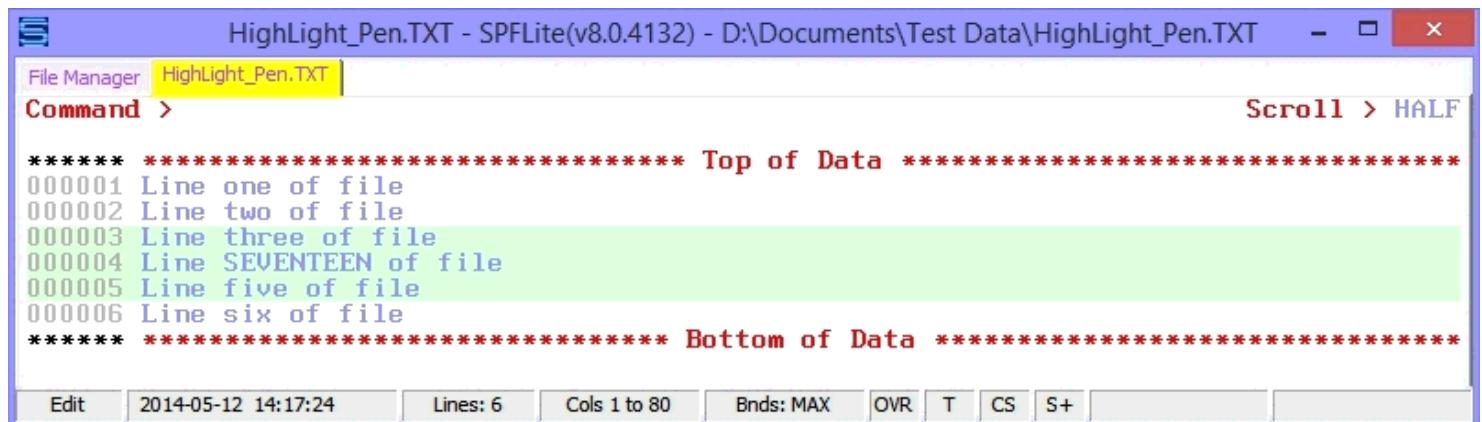
confusing "color clash" situation. As noted above, you **can** mix .AUTO colorization and virtual highlighting pens, but you need to be careful not to confuse yourself.

Example of using highlighting pens

It's easy to highlight text. You will have to establish a mapping for the **(Pen*)** functions. Here is a suggested mapping you may find useful. As always, you are free to map these any way you wish:

Ctrl-Shift-R = [\(Pen/Red\)](#)
Ctrl-Shift-B = [\(Pen/Blue\)](#)
Ctrl-Shift-G = [\(Pen/Green\)](#)
Ctrl-Shift-Y = [\(Pen/Yellow\)](#)
Ctrl-Shift-S = [\(Pen/Std\)](#)

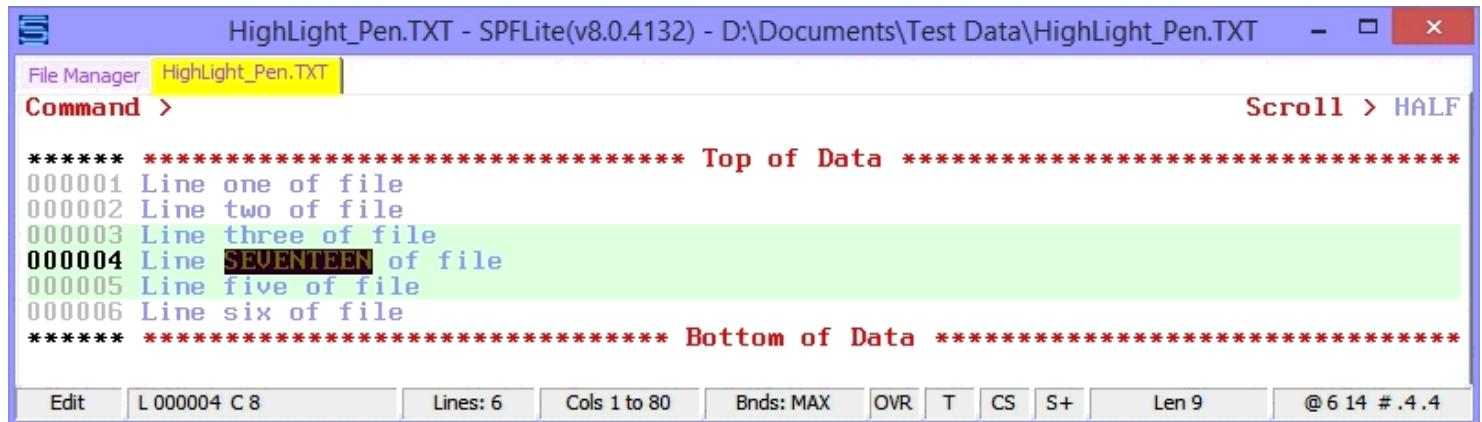
Here is our data file:



```
***** ***** Top of Data *****  
000001 Line one of file  
000002 Line two of file  
000003 Line three of file  
000004 Line SEVENTEEN of file  
000005 Line five of file  
000006 Line six of file  
***** ***** Bottom of Data *****
```

File Manager HighLight_Pen.TXT | Command > | Scroll > HALF | Edit | 2014-05-12 14:17:24 | Lines: 6 | Cols 1 to 80 | Bnds: MAX | OVR | T | CS | S+ | Len 9 | @ 6 14 # .4 .4

We are going to highlight the word **SEVENTEEN**, since it is obviously out of place here. By double-clicking on the word, or by using the Shift+arrow keys, highlight (that is, *select*) the text:



```
***** ***** Top of Data *****  
000001 Line one of file  
000002 Line two of file  
000003 Line three of file  
000004 Line SEVENTEEN of file  
000005 Line five of file  
000006 Line six of file  
***** ***** Bottom of Data *****
```

File Manager HighLight_Pen.TXT | Command > | Scroll > HALF | Edit | L 000004 C 8 | Lines: 6 | Cols 1 to 80 | Bnds: MAX | OVR | T | CS | S+ | Len 9 | @ 6 14 # .4 .4

Finally, while this word is highlighted (again, that is, *selected*), press one of the highlighting-pen keys you mapped above.

Here, we are going to press the Ctrl-Shift-Y key to make the text yellow:

```
***** **** Top of Data ****
000001 Line one of file
000002 Line two of file
000003 Line three of file
000004 Line SEVENTEEN of file
000005 Line five of file
000006 Line six of file
***** **** Bottom of Data ****
```

That's all there is to it.

If you want to "uncolor" this word, you can select it again, and use the key mapped to [\(PenStd\)](#), which was mapped to Ctrl-Shift-S in our example. Or, you can use the **RESET** command, discussed next.

Resetting highlighting pen colors back to the default

RESET can be used to clear highlighting-pen colors from the entire file, or from a given line range. The color name you specify reverts back to the standard text color. So, if you had some text marked up in **BLUE**, and you want that text returned to the normal display color, you would say **RESET BLUE**.

If you want to clear all of the colors back to the default, you can use **RESET STD** or **RESET COLOR**, both of which mean the same thing.

Highlighting pens and special situations

If you do a **FIND** command to find text has been colored with a highlighting pen, the highlighting produced by the **HILITE FIND** profile option will take precedence. Once you find some other string, or do a **RESET** command, the **HILITE FIND** highlighting will go away, and the color from the highlighting pen will be restored.

This of course assumes you have not specified **another** color on the actual **FIND** command itself. If so, the new color specified on the **FIND** command takes effect. Example: **FIND ALL ABCD +GREEN** would set all ABCD strings to GREEN. A subsequent **FIND ALL ABCD +RED** would change the coloring of all ABCD strings from GREEN to RED.

If you swap lines using the **M/MM** and **W/WW** line commands, and any of the lines involved have been colored with a highlighting pen, the colors will be retained when the lines are moved to their new positions.

Be aware that **space characters** in a file are ordinary data, and can have a color associated with them. This can affect **FIND** and **CHANGE** commands and the < and > Data Shifting line commands. See [Shifting Data](#) for more information.

NOTE lines and highlighting pens

At present, you cannot use the the virtual highlighting pen functions to color the text of **NOTE** or **xNOTE** lines.

Working with **SPLIT** and **JOIN** Commands

Contents of Article

[Introduction](#)

[SPLIT Command Examples](#)

[Example 1: Split comma-separated values, discarding commas](#)

[Example 2: Split comma-separated values, retaining commas before split](#)

[Example 3: Split comma-separated values, retaining commas after split](#)

[Example 4: Split data without delimiters](#)

[Example 5: Split using Picture and Format strings](#)

[Example 6: Split on dashes](#)

[Example 7: Split on delimiter and blank](#)

[Example 8: Split using F'!|' change string](#)

[Example 9: Split using a regular expression](#)

[Example 10: Split on left using P'\[string'](#)

[JOIN Command Examples](#)

[Example 11: Join on left](#)

[Example 12: Join on right](#)

[Example 13: Replacement join](#)

[Example 14: Left join with dash inserted](#)

[Example 15: Right join with space between](#)

[Example 16: Right join without space between](#)

Introduction

The **SPLIT** edit primary command is used to selectively split apart lines of text based on a search string. After a split occurs, a line on which the **from-string** is found will become two lines. Everything before the "split point" will be on the first line, while everything after the split point will be on the second line.

The **JOIN** edit primary command is used to selectively combine lines of text based on a search string. After a join occurs, a line on which the **from-string** is found will be combined either with the line before it, or with the line after it. So, each time a join takes place, two lines of text will become one line of text.

This section provides a number of examples of how to use the **SPLIT** and **JOIN** commands. Because these commands provide complementary functions, they are grouped together here. See [SPLIT - Split Lines Using Find/Change Strings](#) and [JOIN - Join lines Using Find/Change Strings](#) for detailed descriptions of the command syntax.

Because the precise state of your data lines may be critical to how a **SPLIT** or **JOIN** command operates, in some cases (depending on your requirements) it may be important to manage the trailing blanks that might exist on the lines you are splitting or joining. For example, if you wished to trim the entire file of trailing blanks, you can place a line command of **TR/** on line 1 and press Enter. You may find the line commands [TR/TRR](#), [PL/PLL](#) and [TL/TLL](#), and the primary commands [APPEND](#) and [PREPEND](#) to be useful in conjunction with **SPLIT** and **JOIN**, either to prepare lines for SPLIT/JOIN processing, or to modify them afterwards.

Bear in mind that **SPLIT** and **JOIN** are treated as specialized forms of **FIND** and **CHANGE**. This means that the **RFIND/RLOCFIND** and **RCHANGE** commands (usually mapped to F5 and F6) may be used to selectively find text on lines to be split or joined using F5, and then you can selectively perform the **SPLIT** or **JOIN** with F6 if that is what meets your requirements; it's not necessary to always use the **ALL** keyword to process your entire line range.

A JOIN **from-string** may be specified as a Regular Expression, in addition to a Picture string.

A JOIN **from-string** must be either:

- a Picture string in one of the following formats:

P' [string]

JOIN is being asked to perform a *left-join operation*. The value of **string** must appear on the left side of lines within the line range in order to be joined; otherwise any lines not beginning with **string** will be ignored for JOIN purposes. When a **[** Picture code appears in the **from-string** of a JOIN command, it must appear in the left-most position of the Picture string, and nowhere else.

P' string]

JOIN is being asked to perform a *right-join operation*. The value of **string** must appear on the right side of lines within the line range in order to be joined; otherwise any lines not ending with **string** will be ignored for JOIN purposes. When a **]** Picture code appears in the **from-string** of a JOIN command, it must appear in the right-most position of the Picture string, and nowhere else.

- a Regular Expression string in one of the following formats:

R' ^expression'

JOIN is being asked to perform a *left-join operation*. The regular expression **must** start with the **^** directive to indicate the left-hand edge of the line. The remaining expression may be any valid RegEx expression.

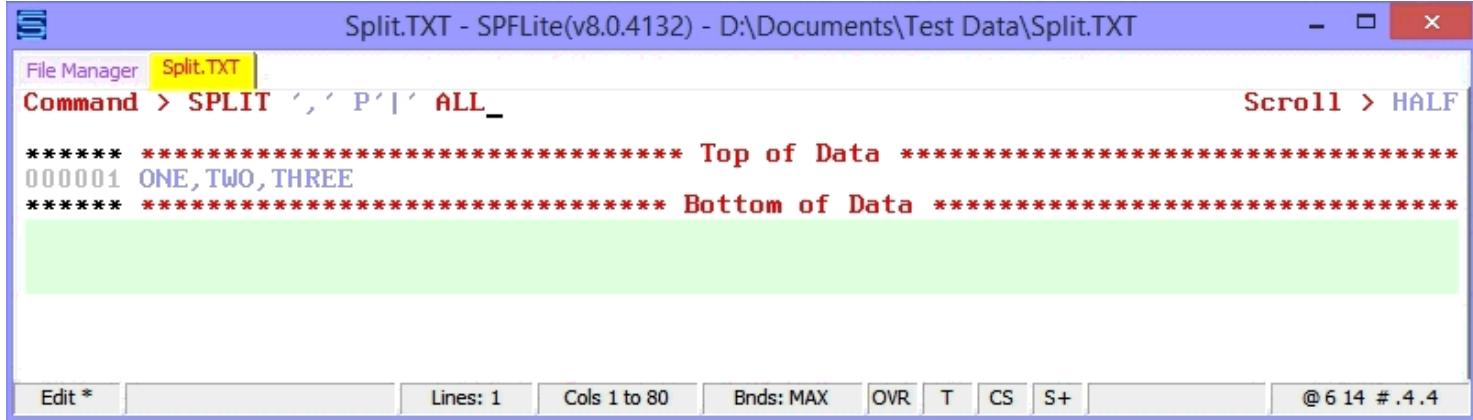
R' expression\$'

JOIN is being asked to perform a *right-join operation*. The regular expression **must** end with the **\$** directive to indicate the right-hand edge of the line. The remaining expression may be any valid RegEx expression.

The section on JOIN command examples starts [here](#).

SPLIT Command Examples

Example 1: Split comma-separated values, discarding commas



The screenshot shows the SPFLite interface with the following details:

- File Manager:** Split.TXT
- Command:** SPLIT **,** **,** P' **|** **ALL**
- Preview Area:** Shows the input data and the resulting split data.
 - Top of Data:** 000001 ONE, TWO, THREE
 - Bottom of Data:** (empty)
- Status Bar:** Lines: 1, Cols 1 to 80, Bnds: MAX, OVR, T, CS, S+, @ 6 14 # .4 .4

Result: Each place that a comma is found is split to form a new line. Two commas are found, so two splits occur. Since the change Picture has the vertical bar | split code but nothing else, the commas that are found are "consumed" by the splitting process, and so they don't appear in the result.

Example 2: Split comma-separated values, retaining commas before split



Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > SPLIT ' ' P',|' ALL_

Scroll > HALF

```
***** **** Top of Data ****
000001 ONE, TWO, THREE
***** **** Bottom of Data ****
```

Edit * Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result: Each place where a comma is found is split to form a new line. Two commas are found, so two splits occur. Since the change Picture has a comma **before** the vertical bar | split code, the commas that are found are first "consumed" by the splitting process, and then re-inserted back because of appearing in the Picture string, so that they appear at the ends of lines 1 and 2.

Example 3: Split comma-separated values, retaining commas after split

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager **Split.TXT** Command > SPLIT ',' P'|,' ALL_ Scroll > HALF

```
***** **** Top of Data ****
000001 ONE, TWO, THREE
***** **** Bottom of Data ****
```

Edit * Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result: Each place where a comma is found is split to form a new line. Two commas are found, so two splits occur. Since the change Picture has a comma **after** the vertical bar | split code, the commas that are found are first "consumed" by the splitting process, and then restored because of appearing in the Picture string, so that they appear at the beginning of lines 2 and 3. **Compare examples 2 and 3.** See how characters **before** the vertical bar are on the **first** line of a split, while characters **after** the vertical bar are on the **second** line of a split.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager **Split.TXT** Command > Scroll > HALF

Split performed 2 times

```
***** **** Top of Data ****
000001 ONE
000002 ,TWO
000003 ,THREE
***** **** Bottom of Data ****
```

Edit * L 000002 C 1 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0004

Example 4: Split data without delimiters

When you have to split some text that doesn't have any delimiters, you may have to repeat some or all of the find-string data in the change string.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager **Split.TXT** Command > SPLIT ONETWO p'ONE|TWO' Scroll > HALF

```
***** **** Top of Data ****
000001 ONETWO
***** **** Bottom of Data ****
```

Edit 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result: ONETWO on one line is replaced by ONE and TWO on two lines. We had to repeat the original data appearing in the change-string with a split mark of | in the middle of the change string.

```

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT
File Manager Split.TXT
Command > SPLIT ONE TWO
***** **** Top of Data ****
000001 ONE
000002 TWO
***** **** Bottom of Data ****
Edit * L 000002 C 3 Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003

```

Example 5: Split using Picture and Format strings

You can use a Picture for the find-string, so that the change-string is handled 'generically'. Here we are splitting all strings of 6 upper case letters. Note the use of the F-type Format string for the change-string.

We use a **Format**, because the '=' signs don't correspond to the same positions between the find and change strings, and so a P-type Picture string won't work. That is, the right-hand 3 '=' signs are in position 5-7 of the Format string, but they correspond to positions 4-6 of the original data.

The one-character shift is the result of the fact that the vertical bar | code does not correspond to a character position of original data, but it 'takes up a place' in the string, so we must use a Format string here in order to be consistent with how Formats are used elsewhere, like in **CHANGE** commands.

If you had used a Picture here instead of a Format, you would get an error message, "**CHANGE chars =<>~ appear past the length of the Find string**". As you can see below, the Format string contains 7 characters (6 = equal signs and one | vertical bar) whereas the find-string is a Picture string of length 6. If the change string were a Picture instead of a Format, the right-most = equal sign would correspond to "position 7" of the found string. Since the first operand is of length 6, there is no position 7, and that's why an error occurs. Format strings deal with character positions differently, and that's why it works. See [Specifying a Picture or Format String](#) for a detailed discussion of these issue.

```

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT
File Manager Split.TXT
Command > SPLIT P'>>>>' F'==|===' ALL
***** **** Top of Data ****
000001 ONETWO
000002 SIXTEN
***** **** Bottom of Data ****
Edit * 2014-05-12 14:46:22 Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+

```

Result: Both lines are split, making four lines from the original two. In the change Format string, the first three = equal signs, before the | vertical bar, correspond to the first three positions (1-3) of the strings ONETWO and SIXTEN, and the second three = equal signs, after the | vertical bar, correspond to the last three positions (4-6) of those strings.

Now you can start seeing some of the real power in using **SPLIT** with Pictures. You are not just finding delimiters and breaking lines apart by using them, but here we actually split lines based upon undelimited text contents.

File Manager **Split.TXT** | Command > **Scroll > HALF**
Split performed 2 times

```
***** **** Top of Data ****
000001 ONE
000002 TWO
000003 SIX
000004 TEN
***** Bottom of Data ****
```

Edit * L 000002 C 3 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003

Example 6: Split on dashes

Split a line on dashes, which are then discarded. Note that a literal | vertical bar in the data does not cause line splits, and is not confused in the **SPLIT** operation. You can also specify the command as **SPLIT ' - ' F' | ' ALL**.

File Manager **Split.TXT** | Command > **SPLIT ' - ' F' | ' ALL** **Scroll > HALF**

```
***** **** Top of Data ****
000001 ONE-TWO-THREE|FOUR
***** Bottom of Data ****
```

Edit * 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result: See that the | literal data is still present. The mere fact that a Picture split code is specified as | has nothing to do with an ordinary | vertical-bar character appearing in your data.

File Manager **Split.TXT** | Command > **Scroll > HALF**
Split performed 2 times

```
***** **** Top of Data ****
000001 ONE
000002 TWO
000003 THREE|FOUR
***** Bottom of Data ****
```

Edit * L 000002 C 1 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003

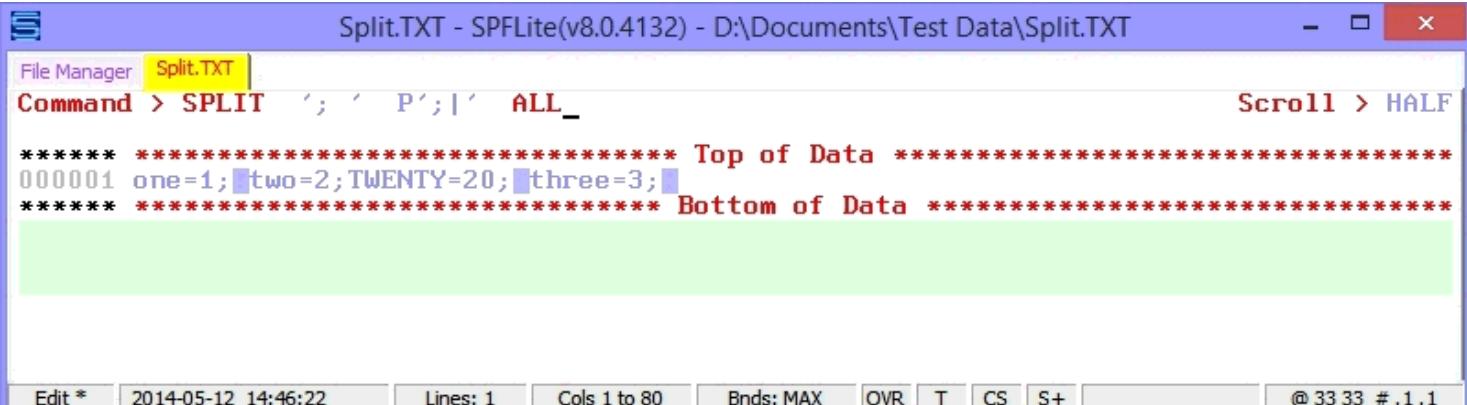
Example 7: Split on delimiter and blank

Split apart some programming statements: only split where a semicolon is followed by a blank, then discard the blank after splitting.

Because split-point strings are discarded, the resulting lines don't have trailing blanks on them. Blanks have been highlighted in **blue** for clarity. The split points are substituted where the blanks were located, and are then

discarded after the split.

You can also specify the command as **SPLIT ' ; ' F' ; | ' ALL**.

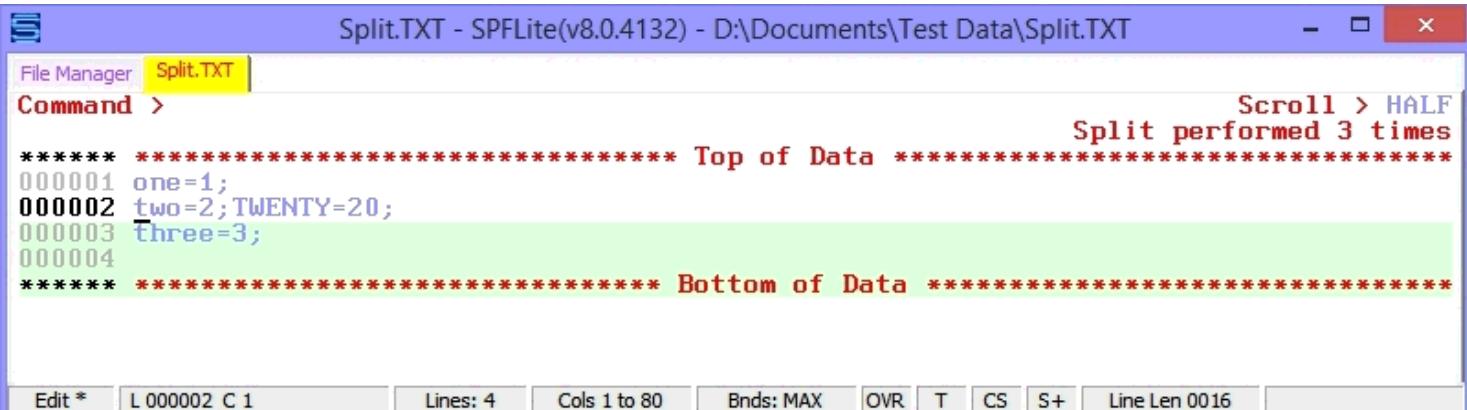


The screenshot shows a text editor window titled "Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT". The status bar indicates "Edit * 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ @ 33 33 # .1.1". The command line at the top shows "Command > SPLIT ' ; ' P' ; | ' ALL_". The text area displays the following content:

```
***** ***** Top of Data *****
000001 one=1;two=2;TWENTY=20;three=3;
***** ***** Bottom of Data *****
```

The line "two=2;TWENTY=20;" is not split, as there is no blank character after the semicolon.

Result: Notice that the part with "two=2;TWENTY=20;" does not get split. That's because the part with "two=2;" didn't have a blank after it, and the SPLIT find-picture insists on a blank after the , semicolon. The result lines do not have any trailing blanks on them, and the last line is a zero-length line.



The screenshot shows a text editor window titled "Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT". The status bar indicates "Edit * L 000002 C 1 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0016". The command line at the top shows "Command > SPLIT ' ; ' P' ; | ' ALL". The text area displays the following content:

```
***** ***** Top of Data *****
000001 one=1;
000002 two=2;TWENTY=20;
000003 three=3;
000004
***** ***** Bottom of Data *****
```

The line "two=2;TWENTY=20;" is split into two lines: "two=2;" and "TWENTY=20;". The status bar shows "Line Len 0016", indicating the last line is empty.

Example 8: Split using **F'!|'** change string

Split apart lines using the ! code in for string-2. This ! code represents the **entire** string found by the 'find-string' of **T'ABC'**, which in this case will be any of the letters ABC regardless of case. Because the entire string that was found will appear in the change-string, no source characters are deleted (lost) during the split process.

This command could also have been specified as **SPLIT T'ABC' P'==|=|' ALL** or as **SPLIT T'ABC' F'==|=|' ALL**, but using the ! notation is more concise, because no matter how long the found-string is, only a single ! code is needed in the change Format.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > SPLIT T'ABC' F'!|' ALL

***** * ***** Top of Data *****
000001 one=ABCtwo=Abcthree=abc
***** * ***** Bottom of Data *****

Edit * 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

Result:

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command >

***** * ***** Top of Data *****
000001 one=ABC
000002 two=Abc
000003 three=abc
000004
***** * ***** Bottom of Data *****

***** Split performed 3 times *****

Edit * L 000002 C 1 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

Example 9: Split using a regular expression

Sometimes your data is not very well-defined, but you need to split it anyway. Below, we have a file in which we want to split lines where there is a "word" at the beginning of the line, so that the split happens at whatever follows the "word". The problem is that the words vary in size, don't always begin the line, and have inconsistent data following them (digits, special characters and blanks appear after the words). There is no specific data we can "home in on" to decide where splits should be done. The best way to find such strings is with a Regular Expression. The Regular Expression codes `^` and `$` correspond to Picture codes of `[` and `]`. We can use that fact to look for "words" that only begin a line and nowhere else. Then, using the fact that the `!` Picture/Format code represents the entire found-string regardless of length, that will do nicely for the change-string.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

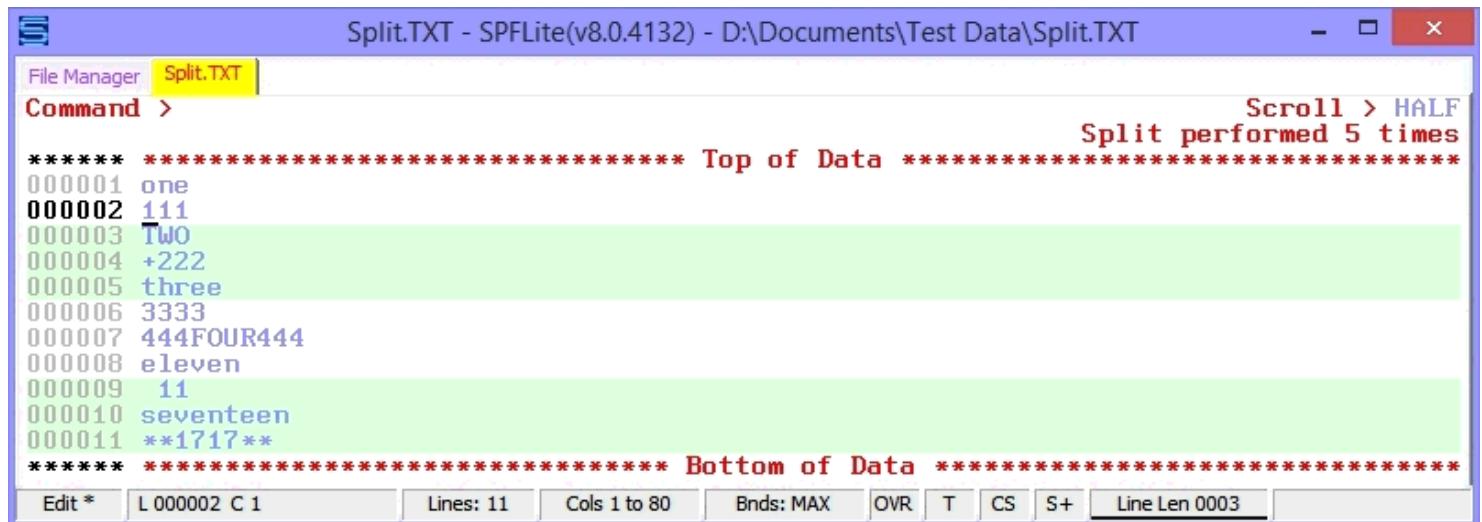
Command > SPLIT R'^[A-Z]+' F'!|' ALL

***** * ***** Top of Data *****
000001 one111
000002 TWO+222
000003 three3333
000004 444FOUR444
000005 eleven 11
000006 seventeen**1717**
***** * ***** Bottom of Data *****

Edit * 2014-05-12 15:11:14 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

Result: Note that on original line 4, the data **444FOUR444** doesn't meet the criteria, because the "word" **FOUR** doesn't start the line. So, that line doesn't get split. Out of 6 original lines, just 5 of them are split. See how the `!` code represents each of the "word" strings in turn, even though they are of differing sizes. This example also

demonstrates how powerful Regular Expressions can be. We needed to find strings that were ended, not by a delimiter or by a fixed length, but by a character that was not in the correct "class" (alphabetic, in this case). Only a Regular Expression could do that.

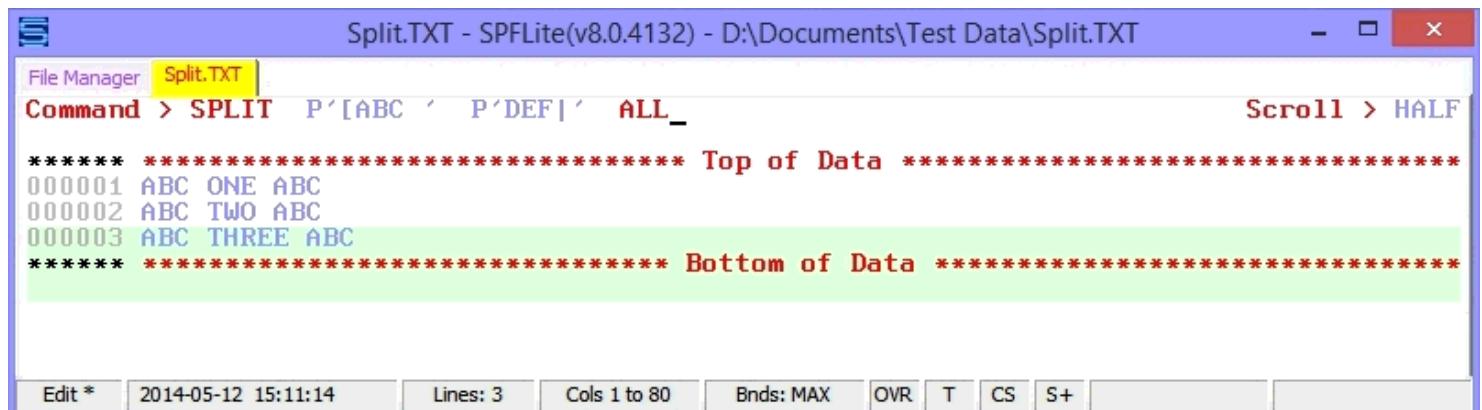


```
***** ***** Top of Data *****
000001 one
000002 111
000003 TWO
000004 +222
000005 three
000006 3333
000007 444FOUR444
000008 eleven
000009 11
000010 seventeen
000011 **1717**
```

```
***** ***** Bottom of Data *****
Edit * L 000002 C 1 Lines: 11 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003
```

Example 10: Split on left using `P'[string]`

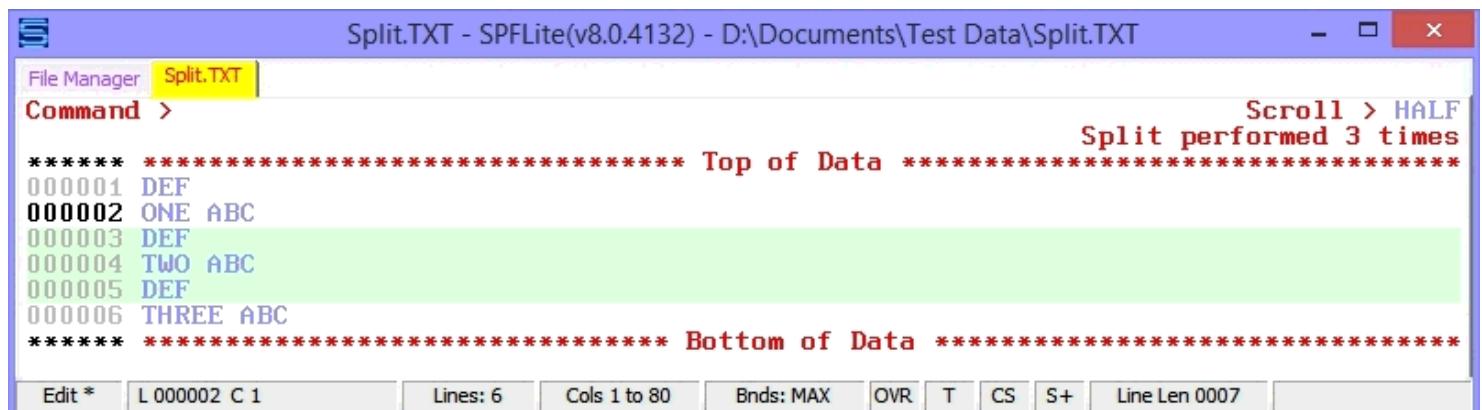
Given the file below, we want to replace the first ABC on each line with DEF, and remove the blank that follows it, and then split the lines.



```
***** ***** Top of Data *****
000001 ABC ONE ABC
000002 ABC TWO ABC
000003 ABC THREE ABC
```

```
***** ***** Bottom of Data *****
Edit * 2014-05-12 15:11:14 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003
```

Result:



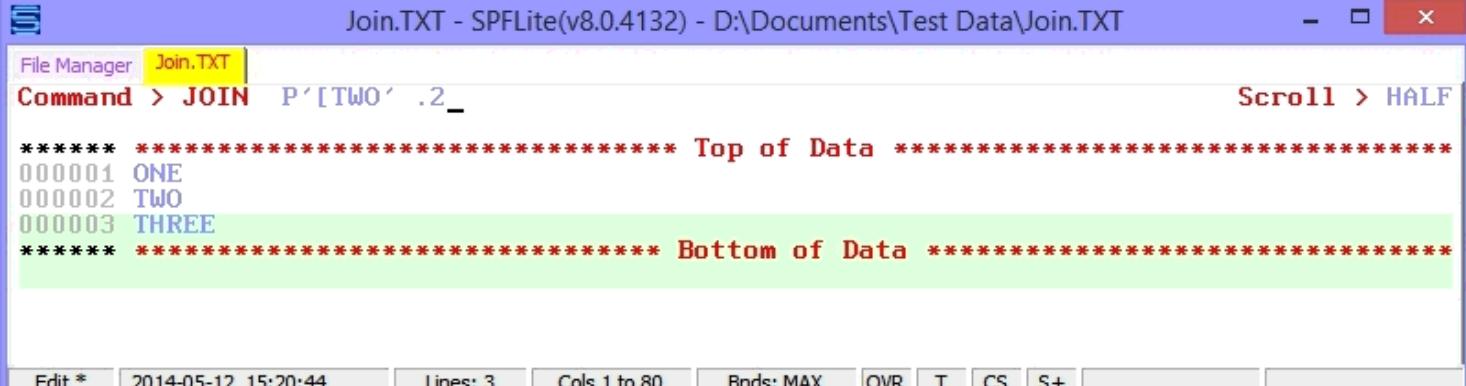
```
***** ***** Top of Data *****
000001 DEF
000002 ONE ABC
000003 DEF
000004 TWO ABC
000005 DEF
000006 THREE ABC
```

```
***** ***** Bottom of Data *****
Edit * L 000002 C 1 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007
```

JOIN Command Examples

Example 11: Join on left

A left-join is performed on line 2.



Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT

File Manager Join.TXT

Command > JOIN P'[TWO' .2_

***** * ***** Top of Data *****

000001 ONE

000002 TWO

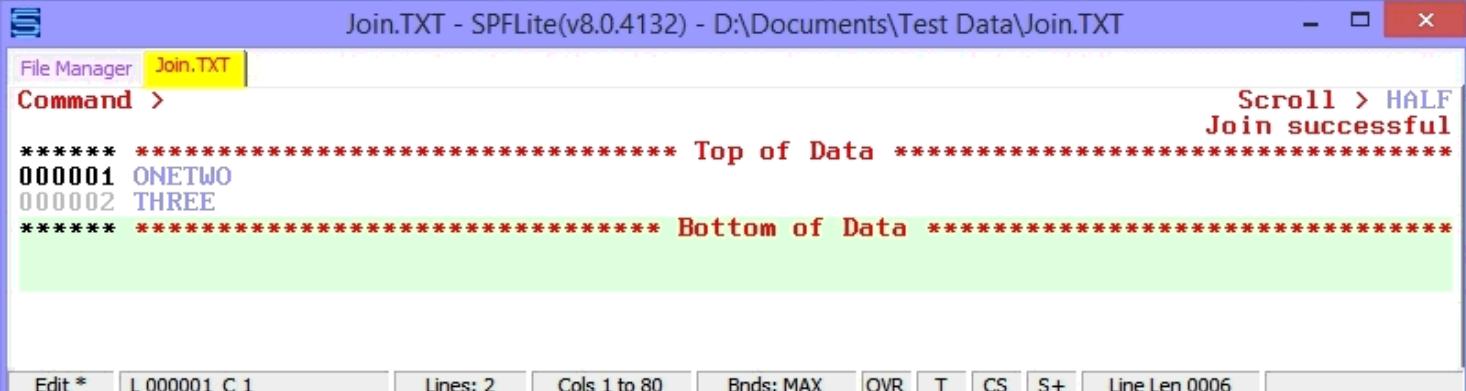
000003 THREE

***** * ***** Bottom of Data *****

Edit * 2014-05-12 15:20:44 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006

This screenshot shows the SPFLite editor with a file named 'Join.TXT'. The command 'JOIN P'[TWO' .2_' is entered in the command line. The output shows the 'Top of Data' (lines 1 and 2) followed by the 'Bottom of Data' (line 3). Line 3 is highlighted in green.

Result: The left side of line 2 is joined with the right side of line 1. Former lines 1 and 2 are combined into a new line 1, and the former line 3 becomes the new line 2.



Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT

File Manager Join.TXT

Command >

***** * ***** Top of Data *****

000001 ONETWO

000002 THREE

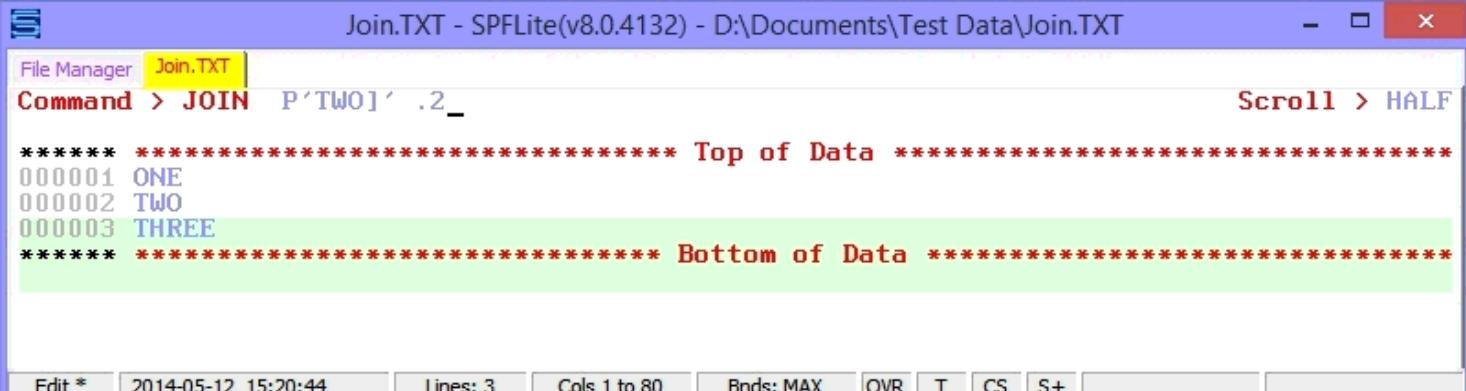
***** * ***** Bottom of Data *****

Edit * L 000001 C 1 Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006

This screenshot shows the SPFLite editor after the join operation. The command line is empty. The output shows the combined line 'ONETWO' followed by 'THREE'. The status bar indicates 'L 000001 C 1'.

Example 12: Join on right

A right-join is performed on line 2.



Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT

File Manager Join.TXT

Command > JOIN P'TWO]' .2_

***** * ***** Top of Data *****

000001 ONE

000002 TWO

000003 THREE

***** * ***** Bottom of Data *****

Edit * 2014-05-12 15:20:44 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0006

This screenshot shows the SPFLite editor with a file named 'Join.TXT'. The command 'JOIN P'TWO]' .2_' is entered in the command line. The output shows the 'Top of Data' (lines 1 and 2) followed by the 'Bottom of Data' (line 3). Line 3 is highlighted in green.

Result: The right side of line 2 is joined with the left side of line 3. Former lines 2 and 3 are combined into a new line 2.

```
File Manager Join.TXT
Command > Scroll > HALF
Join successful
***** **** Top of Data **** ****
000001 ONE
000002 TWO THREE
***** Bottom of Data **** ****
Edit * L 000002 C 1 Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0008
```

Example 13: Replacement join

If you look at the first two **JOIN** examples (11 and 12 above), you will see that there is no second string operand. When you omit this, **JOIN** assumes that the second operand is **p'!'**. That is, whatever string is found using the first operand is appended to the joined line. (Recall that **p'!'** stands for the value of the **entire** found string, **regardless of its length**.) In example 12, this means that you will get the same results by any of the following commands:

```
JOIN P'TWO] ' .2  
JOIN P'TWO] ' 'TWO' .2  
JOIN P'TWO] ' P'!' .2
```

Try it yourself, and convince yourself that this works. We can call this a "simple join".

Now, internally this is what is really happening:

- A line with "TWO" at the end is found (the] right bracket means "at the end")
 - The string "TWO" is deleted from the end of line 2
 - The contents of line 2 (with the ending "TWO" removed) **plus** the string "TWO" **plus** the contents of line 3 are concatenated together to make a new line 2.

Why are we describing this in such excruciating detail? It all seems very obvious, even redundant, that it works this way. Well, not entirely. The point is, we can use something for the "middle" part of this concatenation process **other than** the string that is found.

Here, we are going to join lines 2 and 3 again. But, instead of merely joining them, the string "TWO" is going to be replaced by "222". When you understand the process above for a "simple join", joins that are not as simple will make sense to you.



Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT

File Manager **Join.TXT**

Command > JOIN P'TWO' '222' .2

Scroll > HALF

***** * Top of Data *****

000001 ONE
000002 TWO
000003 THREE

***** * Bottom of Data *****

Edit * 2014-05-12 15:20:44 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ []

Result: As you can see, the string "TWO" that was found by the first operand of JOIN is replaced by "222" instead of being merely repeated. If you like, you can call this a "replacement join" if you want a fancy term for it.

```
Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT
File Manager Join.TXT
Command > Scroll > HALF
Join successful
***** **** Top of Data **** ****
000001 ONE
000002 222THREE
***** **** Bottom of Data **** ****
Edit * L 000002 C 1 Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0008
```

Example 14: Left join with dash inserted

In the file below, line 2 is joined to the end of line 1, even though line 1 is not within the line-control-range specified on the command. This is because the leading P'! code on the search string defines a left-side alignment, and left-sided **JOIN**. For a left-sign join, when the first line of the line range is matched by the **JOIN** find string, a join takes place between the first line of the line range and the line which precedes it (if one exists). In this example, that means that line 2, the beginning of the line range .2 .3, will be joined to the line that precedes is, which is line 1.

In the find-picture, the = equal sign successively matches to the first character of each line, which is, in order, **O**, **T** then **T** again. The change string first inserts a – minus sign, then repeats the character matched by the find-string (**O**, **T** then **T** again), and then finally it joins the lines.

A left-side join joins to the line before it, whereas a right-side join joins to the line after it. So, in this case, line 2 is left-joined to line 1, then line 3 is left-joined to line 1.

```
File Manager Join.TXT
Command > JOIN P'[='] F'[-='] ALL .2 .3
Scroll > HALF

***** **** Top of Data **** *****

000001 ONE
000002 TWO
000003 THREE

***** **** Bottom of Data **** *****

Edit * 2014-05-12 15:20:44 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+
```

Result: The lines in the line range are joined together with dashes between them.

Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT

File Manager **Join.TXT**

Command >

***** **** Top of Data *****

000001 ONE-TWO-THREE

***** ***** Bottom of Data *****

Edit * L 000001 C 1 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0013

Example 15: Right join with space between

Assume in the file below that lines 1-3 are trimmed of all trailing blanks. This is a necessary condition for this example, and can be enforced with a TR line command. The change string includes a ' ; ' semicolon to restore the one being matched against, and a blank for readability. Note that the join process consumes the empty line 4, because we are doing a right-join here.

join.txt - SPFLite(v8.0.4132) - D:\Documents\Test Data\join.txt

File Manager **join.txt**

Command > **JOIN P';' ; ' ALL**

***** **** Top of Data *****

000001 one=1;
000002 two=2;
000003 three=3;
000004

***** ***** Bottom of Data *****

Edit 2014-05-12 16:19:48 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0023

Result: The four lines are joined into one. Because line 3 is joined with line 4, and ' ; ' is inserted between them, and line 4 was a zero-length line, line 1 of the result will have one trailing blank on the end, inserted there as part of the ' ; ' change string.

join.txt - SPFLite(v8.0.4132) - D:\Documents\Test Data\join.txt

File Manager **join.txt**

Command >

***** **** Top of Data *****

000001 one=1; two=2; three=3;

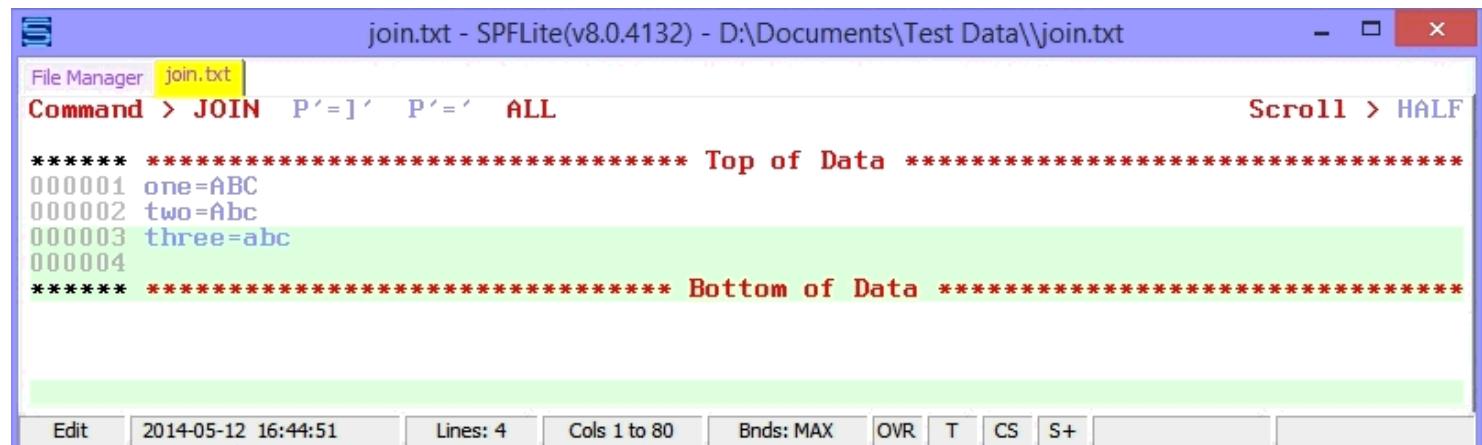
***** ***** Bottom of Data *****

Edit * L 000001 C 1 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0023

Example 16: Right join without space between

Here, we are joining together 3 lines into one with no spaces between. This action consumes line 4. The JOIN

find Picture `P'=[]'` here matches the last character of each line, which is **C**, then **c**, then **c** again. The change Picture of `P'=''` replaces each **C** or **c** "with itself".



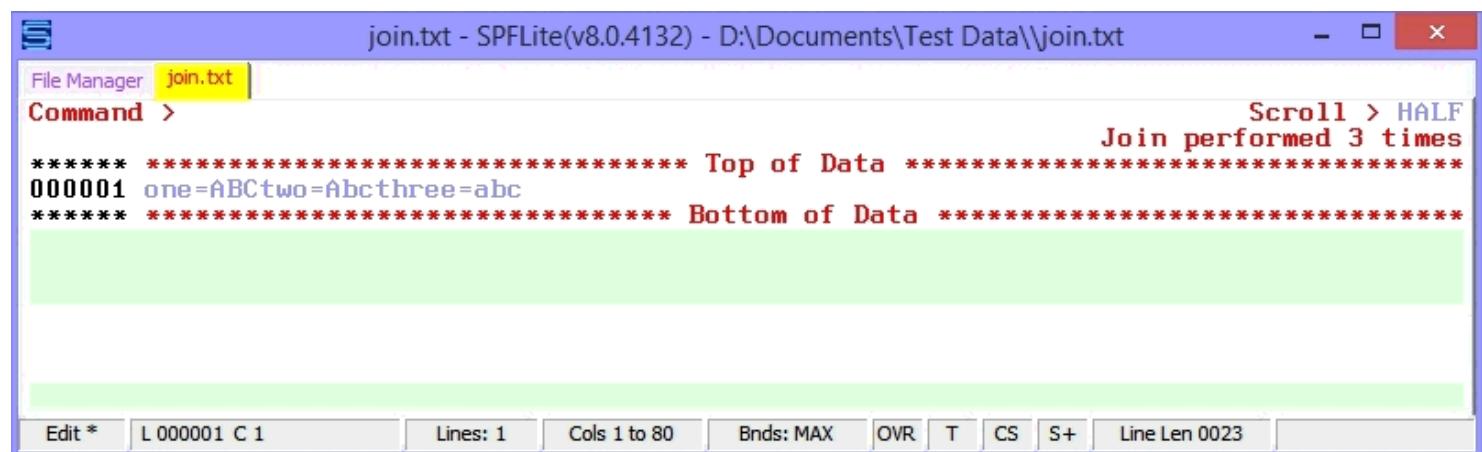
```
join.txt - SPFLite(v8.0.4132) - D:\Documents\Test Data\join.txt
File Manager join.txt
Command > JOIN P'=[ ]' P'=' ALL
Scroll > HALF

***** **** Top of Data ****
000001 one=ABC
000002 two=Abc
000003 three=abc
000004

***** **** Bottom of Data ****

Edit 2014-05-12 16:44:51 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+
```

Result: The three lines are joined together as one, and the blank line 4 has disappeared.



```
join.txt - SPFLite(v8.0.4132) - D:\Documents\Test Data\join.txt
File Manager join.txt
Command >
Scroll > HALF
Join performed 3 times

***** **** Top of Data ****
000001 one=ABCtwo=Abcthree=abc
***** **** Bottom of Data ****

Edit * L 000001 C 1 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0023
```

Performing Searches with Find in Files

Contents of Article

[Introduction](#)

[Syntax](#)

[Example](#)

Introduction

The **Find in Files** command **FF** is issued from the File Manager, and is used to search all currently displayed file names for a string value. You can search a directory list, the **Recent Files** File List or any other File List.

The **FF** command searches every file that is displayed in the current list. You have a number of options to manage and selectively search File Lists.

- To search a 'selective' list of files, you can start with a directory list and use the **File Patterns** field with one or more wildcards to reduce the number of displayed files. You can also add files to a favorites list with the **A** or **ADD** line command or the **FAV** edit primary command, and then issue the **FF** command while the **Favorite Files** List is displayed.
- If you search a File List, you can remove files from the list before issuing the **FF** command, to avoid finding a string in a given file in three ways:
 - with the Forget line command **F**. Keep in mind that if you do this, the file will be permanently removed from the File List. This only removes a name-entry in the File List, but does **not** delete the file itself. (A **FORGET** can be reversed using the [RESET F](#) command for the File List)
 - with the Exclude line command **X**. Excluded lines are hidden only for the current display of this File List, unlike **F** Forgotten files. (An **EXCLUDE** can be reversed using the [RESET X](#) command for the File List)
 - with the File Manager Primary command [EXCLUDE / X](#), which allows you to Exclude files based on a File Pattern operand, handy for bulk excludes of particular format file names.
- If an existing File List is close to what you want, but not exactly, you can **Clone** a File List, which allows you to edit the list of file names it contains, and then save that File List under a new name. You then would click on **Named Favorites** to see the new File List you just created. You might wish to clone a File List if you wish to tailor the list of files it contains without permanently deleting them.
- If a given directory display has the files you are interested in searching, and you wish to search this list repeatedly, you can issue a [MAKELIST](#) command to create a named-favorites File List. Remember that **MAKELIST** can either create a fixed list of files that are currently displayed on a directory list, or it can create a Symbolic "Shortcut" file list using the **SYM** keyword option.

If the **FF** command finds at least one matching file, it creates (or updates) the **Found Files** File List and then displays it.

You can issue the **FF** command starting from the **Found Files** File List **itself**, to search files that were found by a prior **FF** command. If you do this with different search strings, you will successively refine the list. That is, if you issue **FF ABC**, then **FF DEF**, then **FF XYZ**, you will end up with a new **Found Files** File List that only shows files that contain all three strings ABC, DEF and XYZ. No matter how many times you use the **FF** command, there is only one **Found Files** File List.

The Find in Files command **FF** uses a syntax similar to the editor **FIND** command.

Note: The new **FF** alias for the edit **FIND** command should not be confused with the File Manager Find in Files command **FF**, which is completely different, and has nothing to do with the Edit **FIND** and **FF** commands.

Syntax

```
FF string [ start-col [end-col] ] [ CHARS | WORD | PREFIX | SUFFIX ]  
[ NF ]
```

Operands

string

Any string value accepted in an edit session is permitted here. The *string* may be unquoted, or quoted without a string type, or may be quoted with a string type of **C**, **T**, **X**, **P** or **R**.

Unquoted strings, or string quoted without a string type, will be treated as either **C** or **T**, depending on the **Case C/T** code that appears on the SPFLite status line. The **Case C/T** code for the File Manager can be changed by the **CASE C/T** command, as in an edit tab.

start-col

Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.

end-col

Right column of a range (with start-column) within which the search-string value must be found.

CHARS | WORD | PREFIX | SUFFIX

Specifies the 'search context' for the string, the same as is done for the **FIND** command in the editor.

If not specified, a **CHARS** search is done if the status line shows **C** or **T**, and a **WORD** search is done if the status line shows **C W** or **T W**. These two defaults can be set by the [FIND CHARS](#) and [FIND WORDS](#) commands, as in an edit tab.

NF

If the **NF** option is used, a search is made for files that do **not** have the *string* present on any line.

Example

Let's say you have three files in a **Named Favorites** File List called **PROJECT**, and you want to know which files have the string ABC in them. Click on the File Manager tab and then on the line with **PROJECT** File List. You will see a display something like this:

PROJECT - FILELIST - File Manager - SPFLite(v8.0.4133)

File Manager | Command > Scroll > CSR

New FilePath Recent Found Opened Favorites Lists Paths Profile

FILELIST > PROJECT

Cmd	Name +	Size	Date
-----	D:\Documents\Test Data\FILE1.TXT	89	2014-05-10 14:17
-----	D:\Documents\Test Data\FILE2.TXT	111	2014-05-10 14:17
-----	D:\Documents\Test Data\FILE3.TXT	79	2014-05-10 13:38
-- End of List, 3 Files --			

Line Commands: Add to favorites Browse Clone Delete Edit Exclude
 Forget Job submit MultiEdit Print Purge Rename Select Touch
 File Patterns: Masks separated by ; such as *.BAS;*.TXT;*.INC;*.H

SELECT OVR T

Assume that only FILE2.TXT has the string you are looking for. You would issue a File in Files command:

FF ABC

You will see a popup window while the search is going on. When finished, the popup will go away, and you will see a display like this, as SPFLite now displays the **Found Files** File List:

PROJECT - FILELIST - File Manager - SPFLite(v8.0.4133)

File Manager | Command > Scroll > CSR

New FilePath Recent Found Opened Favorites Lists Paths Profile

FILELIST > Found Files: ff abc

1 file satisfied search.

Cmd	Name +	Size	Date
-----	D:\Documents\Test Data\FILE2.TXT	111	2014-05-13 11:31
-- End of List, 1 File --			

Line Commands: Add to favorites Browse Clone Delete DIR Edit Exclude
 Forget Job submit MultiEdit Print Purge Rename Select Touch WDI
 File Patterns: Masks separated by ; such as *.BAS;*.TXT;*.INC;*.H

SELECT OVR T

You can now use the **Found Files** File List to issue any File Manager line command, or click on the name to open the file for editing.

Command Keys and Substitution Strings

SPFLite did not initially support direct interfacing with external scripting engines for macro support, although scripts could be launched using the **CMD** commands. Because of this, a simplified variable substitution system was used to assist in creating custom **CMD** commands, and for key-mapped key definitions, **PRINT SETUP** strings, and **SUBMIT** codes.

Now, with full macro support available, the need for this variable substitution is gone, but support is being retained until such time as it has been superseded.

Substitution Strings are values identified as a prefix of ~ tilde or ^ circumflex followed by a single letter code. The letter code is case-insensitive and can be either upper or lower case; we show upper case here for clarity.

- A ~ tilde prefix before the letter code identifies a Substitution String, and requests that its string value be substituted as-is at the point it appears.
- A ^ circumflex prefix before the letter code identifies a Substitution String, and requests that its string value be substituted in UPPER-CASE at the point it appears. Only the value of the Substitution String is put into upper case, the original data remains as-is.

Currently, the following Substitution Strings are available:

~F and **^F**

This variable will be replaced by the current filename being edited. e.g. If editing C:\MyData\Letters\May01.txt, the ~F variable would become May01.

~K(function) and **^K(function)** **~K(keyname)** and **^K(keyname)** **~K(modifier-keyname)** and **^K(modifier-keyname)**

Here,

- *function* is any primitive keyboard function in parentheses, such as [\(Date\)](#)
- *keyname* is the “official” name of any key in the KEYMAP GUI as described with the “Current Key:” label.
- *modifier* is an optional list of one to three letters from the list of **C**, **A** and/or **S** to indicate Ctrl, Alt and/or Shift. When modifier is used, the letters C, A and/or S may each appear at most once for each one, may appear in any order, and are case-insensitive. The modifier codes are separated from the keyname by a – dash character.

Examples:

- ~K(Date)**
- ~K(F2)**
- ~K(A-F1)**

When the function notation is used, the listed function is used as-is, as if typed from a key-mapped key. Only functions or keys that emit text, such as [\(Date\)](#) are permitted.

When the keyname or modifier-keyname notation is used, the corresponding key mapping for that particular key (as modified) is used to fetch the defined list of functions, text, etc. Any characters in [] brackets or text which is unenclosed is treated as emitted text.

~L and **^L**

Similarly to ~W, SPFLite will assign the contents of the current line on which the cursor is sitting to the ~L (line) variable.

~P and ^P

This variable will be replaced by the current path to the file being edited. e.g. If editing C:\MyData\Letters\May01.txt, the ~P variable would become C:\MyData\Letters.

~S(name) and ^S(name)

This variable will be replaced by the current contents of the defined SET variable called *name*. It is an error if the SET variable called *name* is undefined. ~S returns the value of the variable as-is, and ^S returns the value in upper-case.

~W and ^W

Each time the Enter key, or any other key mapped to issue a primary command is pressed, SPFLite will attempt to extract, if possible, the 'word' on which the cursor is sitting. This 'word' is made available as the substitution string (~W). This enables a mapped key definition to incorporate the 'word' into the stored command.

e.g. **X ALL; F ALL ' ~W'**

If this were assigned using [KEYMAP](#) to the Ctrl-F key then this could be used as follows:

- Place the cursor on a word or string of interest
- Press Ctrl-F
- The display will now show only lines containing the word of interest, all other lines will be in Excluded status.

Similarly,

FIND FIRST ' ~W'

would, following the same instructions, scroll the screen to the first occurrence of the 'word' under the cursor.

~X and ^X

This variable will be replaced by the current file extension of the file being edited. e.g. If editing C:\MyData\Letters\May01.txt, the ~X variable would become txt.

Handling Non-Windows Text Files

Contents of Article

[*Introduction*](#)

[*Common settings for non-Windows files*](#)

[*Determining file attributes through experimentation*](#)

[*EBCDIC files*](#)

[*Custom Translation Tables*](#)

[*Using the EOL settings AUTO and AUTONL*](#)

[*Handling files with lone CR characters*](#)

Introduction

We are normally unconcerned with the file format of common text files or source programs, since most Windows functions and tools naturally work with this format, which is simple variable-length text. That format is often compatible with other, non-Windows systems, but not always.

Occasionally you will attempt to open a file from another system, only to get a disorganized or unreadable screen display - so you know something is wrong with the data's content or format, or with SPFLite's understanding of the file. The problem could be one of several factors:

Record Format There are three basic formats used to organize how multiple text lines are stored in a file. This setting is specified for the Profile variable by using the [RECFM](#) command. They are:

U - Undefined

Undefined is the norm for Windows text files; the individual text lines may be any length and are separated by unique delimiter characters. See Line Delimiters below.

F - Fixed

Fixed indicates that all text lines are the same length, known as the Logical Record Length. See Logical Record Length below. Note that Fixed records **may** be stored with or without Line Delimiters as well.

V - Variable

Variable indicates the text lines may be of any length and are **not** separated by unique delimiters, but are written with a prefix field for each line which indicates the length of that particular line. The prefix field is known as an **RDW** for Record Descriptor Word.

VBI - Variable Big-Endian

Variable Big-Endian indicates the text lines may be of any length and are **not** separated by unique delimiters, but are written with a prefix field for each line which indicates the length of that particular line. The prefix field is known as an **RDW** for Record Descriptor Word. **VBI** specifies an RDW of 4 bytes in **big-endian** format containing the length of the data record **not** including the RDW itself.

VLI - Variable Little-Endian

Variable indicates the text lines may be of any length and are **not** separated by unique delimiters, but are written with a prefix field for each line which indicates the length of that particular line. The prefix field is known as an **RDW** for Record Descriptor Word. **VLI** specifies an RDW of 4 bytes in **little-endian** format containing the

length of the data record **not** including the RDW itself.

Line Delimiters	<p>Windows text files typically separate text lines from each other by using a pair of special characters, the CR (Carriage Return) and LF (Line Feed) characters, having the hex value X'0D0A'. This pair is normally referred to as CRLF for short.</p> <p>Unix, Linux and newer Macintosh systems use just the LF character. Other systems (in older Macintosh systems and some few others) may use just the CR character, but that usage is rare.</p> <p>Some systems even have their own 1 or 2 byte unique delimiter strings. You might also wish to temporarily define your own line delimiters. For example, it could be convenient to temporarily treat a comma as the end of the line if you had a type of file called a CSV, or Comma-Separated-Value.</p> <p>SPFLite allows you to process all these types (including files without delimiters) by setting the Profile option EOL to the requirements of your file.</p> <p>There are also files which are not even consistent within themselves, seemingly using CRLF, LF, FF, and CR combinations in weird combinations. An example of this are SYSOUT files from mainframe emulators such as Hercules. SPFLite addresses such files by providing the EOL types of AUTO and AUTONL, discussed later in this article.</p>
Character Set	<p>Even though the Line delimiter (EOL) may be correct, the data in a text file may be using some other character set encoding technique. If the data seems to be total gibberish, or perhaps has a few extraneous characters at the beginning that don't seem to belong, this could be the cause. SPFLite supports the following character sets via the Profile SOURCE variable.</p>

Note regarding Unicode

The Unicode support in SPFLite is quite limited. SPFLite is **not** a true Unicode text editor, as all editing functions take place internally in native ANSI (Windows MS-1252) mode. The SPFLite support is provided to allow reading in and writing Unicode files which **only contain characters that map to normal ANSI characters**. This may sound very restrictive, but many Unicode files are stored in UTF format because web servers demand that format, **not** because there is a true requirement for the additional (non-ANSI) characters supported in Unicode. For this type of application, the SPFLite support is perfectly adequate.

If you truly need the ability to edit the **full** Unicode character set, then SPFLite will **not** handle your requirements.

The following are supported:

ANSI	This is the default and is the normal Windows character set. ANSI is also known as MS-1252, which is a superset of ISO-8859-1 and the first 256 bytes of Unicode.
UTF8	This format of encoding (one of the Unicode types) is the most 'economic' as the base ANSI characters are stored as 8 bit characters, and only special characters (above ANSI 127 are encoded as larger values.
UTF16 UTF16LE	Another Unicode format, this format of encoding uses 16 bits for each character. There are two types of UTF16, LE (Little Endian) and BE

(Big Endian) which refer to the byte order of the encoded 16 bit value. The normal value for Windows based systems is UTF16/UTF16LE, since the Intel processors that run Windows are Little-Endian devices.

UTF16BE And another Unicode format, this one uses 16 bits for each character. The encoded 16 bit value is stored in Big-Endian format. UTF16BE is used on IBM mainframe systems like z/OS when they process Unicode data as 16-bit values.

EBCDIC This is the standard 8-bit encoding used by IBM mainframe systems. Some further information on EBCDIC is at the end of this article

Logical Record Length When the Record Format is set to **F** (for fixed), the length of the fixed record must be specified using the [LRECL](#) command. The **LRECL** is set to **0** (zero) for conventional Windows variable-length records that are terminated by a CRLF pair.

If you need to enforce a **minimum** logical record length that is greater than zero, see [Managing Line Lengths](#) and [MINLEN - Set Minimum Record Length](#) for more information.

Common settings for non-Windows files

What should you specify when something is obviously not right? It is best to check with the provider of the file, or determine the type of system on which it was created.

If it was a Unix/Linux system or newer Macintosh, **RECFM U, EOL LF, SOURCE ANSI, LRECL 0** then try the following:

If it was an IBM mainframe, then try either:

RECFM V, EOL NONE, SOURCE EBCDIC, LRECL 0
or
RECFM F, EOL NONE, SOURCE EBCDIC, LRECL nn
where **nn** is a record length based on what the file usage appears to be.

If it is a web document such as HTML, then try: **RECFM U, EOL CRLF, SOURCE UTF8, LRECL 0**

Determining file attributes through experimentation

If you lack exact information about the file's format, it may become a matter of trial and error to resolve. Sometimes it will require loading as a simple normal text file and carefully examining the data. It is sometimes helpful here to set SPFLite to use a good ANSI font such as **Raster** (included in the optional Font Package on the web site) since it will correctly show line delimiter control characters like CR and LF in a visible format, making it easier to work out what you're looking at.

For very difficult file issues, you may need to exit SPFLite and examine the data with a Hex Editor. Several free versions are available on the Internet. One such hex editor may be found at <http://www.catch22.net/>

When playing with these parameters trying to find the correct settings, it is sometimes helpful to rename the problem file to a unique file type (e.g PROBLEM.TRYIT) so that you don't interfere with the options for a currently valid file profile.

Be sure to issue a **PROFILE UNLOCK** on this test profile. Then you can simply **EDIT** the file, examine it, alter one or more of the Profile variables you are experimenting with, **CANCEL** out and **EDIT** it again to try the new settings. You can repeat this in a trial-and-error approach, or see if the originator of the file has documentation on the file's format.

Note: Once you 'get it right' remember that SPFLite will, using the same Profile, now successfully write files in this format. For example, say you have a profile for a file type of **EBC** (for EBCDIC). You can now 'convert' a normal Windows text file to this format by simply editing the Windows text file, entering **PROFILE USING EBC**, and then using **CREATE/REPLACE** to write the new file. Then **CANCEL** out of the edit session. Make sure the original file's Profile is **LOCKED** before doing this to prevent making the **USING** permanent.

EBCDIC files

SPFLite internally handles all data as Ansi characters. This is equivalent to the Windows 1252 character set, which is a superset of the first 256 characters of Unicode.

Note: IBM has its own ideas about code pages. What Microsoft calls Windows 1252 isn't as simple to IBM. The reason is that 1252 has changed over the years, most recently to add the Euro character. Even though the exact characters present in this code page have changed, Microsoft still calls it 1252. However, IBM considers "1252" to mean a Windows code page 1252 of the past, prior to the advent of the Euro and some other changes they made. IBM considers the Windows 1252 code page of today to be called **5348**. They take this position because they have to support things like DB/2 databases, where database administrators have to know precisely what data **is**, and is **not**, present in CHARACTER database fields, and the old vs. new 1252 are just **not** the same thing. For the record, SPFLite's idea of 1252 is the same as IBM's idea of 5348. That is, we support the **current** Windows 1252 code page definition of today.

You can also edit EBCDIC files, by setting up a PROFILE for a given file type (that is, a file name extension) that has **SOURCE EBCDIC** associated with it. Only Ansi characters are displayed on the edit screen.

In order for SPFLite to handle EBCDIC data, it must translate it from EBCDIC to Ansi while editing, and from Ansi back to EBCDIC for storing externally. To do that, a translation table is required. SPFLite has already defined such a table, and normally there is nothing you need to do, but the following just explains the process.

Presently, only one translation table is supplied with SPFLite, which converts between Windows 1252 (Ansi) and IBM EBCDIC code page 1140. Code page 1140 is a modern code set, comprised of an earlier code page 037 plus the Euro character. Page 1140 is commonly used in North America for nearly all IBM z/OS mainframe installations. The particular tables used by SPFLite are two-way lossless tables that are based on published IBM code-table documentation. Any otherwise unallocated characters have a unique one-to-one translation, so that no data will be lost or mistranslated while editing the Ansi version of your EBCDIC data, even for "binary" data. (Even the "unallocated characters" have lossless translations based on IBM specifications; we did not "make up" any rules for this just for SPFLite.) If for any reason this table is not suitable for your use (if you need EBCDIC national characters outside the North American and/or European characters in Code Page 1140) it is possible to provide your own table. See [Custom EBCDIC translation tables](#) below.

This default table performs a translation which is identical to the translation performed in the Hercules mainframe emulator when using a configuration file parameter line of **CODEPAGE 1252/1140**.

SPFLite does not dictate how lines are terminated in EBCDIC files. You decide how you want this to be handled. All the existing CR/LF combinations can be used with EBCDIC, and their EBCDIC equivalents are used to terminate lines. SPFLite also supports the EBCDIC New Line character NL = X'15' as an EOL value. Additionally, you may use the non-standard file formats noted below. Users of the Hercules mainframe emulator who need to edit EBCDIC files may find cases where fixed-length files and EOL NONE is required.

There is nothing to prevent you from specifying **EOL NL** in an **ANSI** file. However, the ANSI equivalent of EBCDIC X'15' is X'85', which is not a standard ANSI text delimiter, and so NL may be of limited usefulness outside of EBCDIC files.

Custom Translation Tables

Note: Translation tables.

- File extension is **.SOURCE**
- The default file name for EBCDIC is **EBCDIC.SOURCE**

- More than one translation table can exist at the same time besides **EBCDIC.SOURCE**
- Translation tables need not translate between ANSI and EBCDIC. They may be used to translate between different variants of ASCII code pages, such as between ASCII 437 or ASCII 850 and ANSI 1252.
- The new format of the translation tables is much different

The format of the table over predecessor versions has a number of benefits. One of them is that when a translation table is in Round-Trip/Lossless mode, only one 'side' of the translation table is needed, since the two 'halves' are like mirror images of each other anyway. When SPFLite knows you have such a table, it validates that the values in the table are consistent with that. That means, for each of 256 possible character values in a table, each one must appear once and only once for the table to be valid.

Independent of SPFLite, software is being written to directly generate SPFLite translation tables from IBM-provided code page data, known as **UCM files**. This software is in development and will be made available when ready. Once this new software is ready, a full discussion of all these issues will be documented.

Meantime, to get an idea about the new **.SOURCE** format, here is an example of what the **TxtToSource.exe** conversion program does to the old **EBCDIC.TXT** file, showing the key features of the new format:

```
TT  TITLE='SPFLITE TRANSLATION TABLE'  MODE=RT
TT  GENDATE='2013-11-29 14:36:46'

**  SOURCE file was created from conversion of 'EBCDIC.TXT'

**  AE comment:  ASCII 1252 => EBCDIC 1140
**  EA comment:  EBCDIC 1140 => ASCII 1252

**  _0  _1  _2  _3  _4  _5  _6  _7  _8  _9  _A  _B  _C  _D  _E  _F  EA
0*  00  01  02  L  œ  09  †  7F  97  8D  8E  0B  0C  0D  -  -  |  0*
1*  û  ..  08  ≠  û  '  ˜  %  Š  <  œ  05  06  07  2*
2*  x  81  ,  f  "  0A  û  ^  Š  <  œ  05  06  07  2*
3*  90  `  û  "  "  •  -  û  ~  Š  >  14  15  ź  1A  3*
4*  A0  á  ä  à  á  á  á  ç  ñ  ç  .  <  (  +  |  4*
5*  &  é  ê  ë  è  í  í  í  ï  ï  !  $  *  )  ;  ˜  5*
6*  -  /  Á  Ä  À  Á  Á  Ä  Å  Ç  Ñ  |  ,  %  -  >  ?  6*
7*  ø  É  È  È  È  Í  Í  Í  Í  Í  :  #  @  -  =  "  7*
8*  Ø  a  b  c  d  e  f  g  h  i  <<  >>  ð  ý  þ  ±  8*
9*  °  j  k  l  m  n  o  p  q  r  a  °  æ  .  è  9*
A*  µ  ~  s  t  u  v  w  x  y  z  i  ð  ð  ý  þ  ®  A*
B*  ^  £  ¥  .  ©  S  ¶  ¼  ½  ¾  [  ]  -  ..  '  x  B*
C*  {  A  B  C  D  E  F  G  H  I  ò  ö  ò  ö  õ  C*
D*  }  J  K  L  M  N  O  P  Q  R  1  û  ü  û  ü  ú  ü  D*
E*  \  ÷  S  T  U  V  W  X  Y  Z  2  Õ  Ö  Õ  Ö  Õ  Ö  E*
F*  0  1  2  3  4  5  6  7  8  9  3  Ú  Ü  Ú  Ü  Ú  Ü  F*

EA  _0  _1  _2  _3  _4  _5  _6  _7  _8  _9  _A  _B  _C  _D  _E  _F  EA
0_  00  01  02  03  9C  09  86  7F  97  8D  8E  0B  0C  0D  0E  0F  0_
1_  10  11  12  13  9D  85  08  87  18  19  92  8F  1C  1D  1E  1F  1_
2_  A4  81  82  83  84  0A  17  1B  88  89  8A  8B  8C  05  06  07  2_
3_  90  91  16  93  94  95  96  04  98  99  9A  9B  14  15  9E  1A  3_
4_  20  A0  E2  E4  E0  E1  E3  E5  E7  F1  A2  2E  3C  28  2B  7C  4_
5_  26  E9  EA  EB  E8  ED  EE  EF  EC  DF  21  24  2A  29  3B  AC  5_
6_  2D  2F  C2  C4  C0  C1  C3  C5  C7  D1  A6  2C  25  5F  3E  3F  6_
7_  F8  C9  CA  CB  C8  CD  CE  CF  CC  60  3A  23  40  27  3D  22  7_
8_  D8  61  62  63  64  65  66  67  68  69  AB  BB  F0  FD  FE  B1  8_
9_  B0  6A  6B  6C  6D  6E  6F  70  71  72  AA  BA  E6  B8  C6  80  9_
```

<u>A</u>	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	DD	DE	AE	A
<u>B</u>	5E	A3	A5	B7	A9	A7	B6	BC	BD	BE	5B	5D	AF	A8	B4	D7	B
<u>C</u>	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5	C
<u>D</u>	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF	D
<u>E</u>	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5	E
<u>F</u>	30	31	32	33	34	35	36	37	38	39	B3	DB	DC	D9	DA	9F	F

// _0 _1 _2 _3 _4 _5 _6 _7 _8 _9 _A _B _C _D _E _F //

Using the EOL settings AUTO and AUTONL

The End of Line profile options AUTO and AUTONL allow for automatic detection of line terminations, possibly containing inconsistent and spurious line terminators, so that files edited across different system, mainframe SYSOUT files, and other inconsistently-terminated text files can be opened, viewed and edited in a reasonable way. **EOL AUTO/AUTONL** may be applied to non-mainframe files as well, to handle situations where a file's line termination is inconsistent for some reason. A possible cause of this is a file shared between Windows and Unix on a network and edited with different editors that apply different line endings. Line terminations under **EOL AUTO/AUTONL** are handled as follows:

- FF (form feed) characters delimit lines, and cause a =PAGE> marker to be placed in the sequence area. Notes:
 - Scrolling commands **PAGE UP** and **PAGE DOWN** will locate these marked lines.
 - Since **PAGE UP** and **PAGE DOWN** will move the file to these =PAGE> marker lines, which may have a variable number of lines involved, to regain the 'full screen motion' that **PAGE UP** and **PAGE DOWN** does in other files, you can use a scroll amount of **HALF** or **DATA**, or you could enter a numeric value for a specific number of lines. For most users who would have used **PAGE UP** and **PAGE DOWN**, scrolling **UP/DOWN** by the **DATA** scroll amount should work well for them.
 - When you have a file that shows this =PAGE> marker on a line, and you **PRINT** this file, you will have a Form Feed sent to the printer for every line containing the =PAGE> marker.
- A lone LF (line feed) is treated as a line delimiter equivalent to CR,LF
- "Spurious" CR characters that seemingly don't belong there, such as CR,CR,LF are ignored. For example, in the sequence CR,CR,LF, the first CR is spurious; the remaining CR,LF pair is a normal line termination.
- ACR,FF or CR,LF,FF sequence is considered as the end of one line, followed by a page separator line.
- A hex value of X'1A' at the end of the file is ignored.

PAGE Profile Support

An extension to the **AUTO / AUTONL** support is the **PAGE** Profile option. When selected (ON) and an **AUTO / AUTONL** file is processed, the screen display will, when fewer lines exist on a page than the screen height, leave the bottom of the screen page blank rather than display the beginning lines of the next page. This presents a more normal 'print page' format for viewing.

If only **UP PAGE** and **DOWN PAGE** commands are used to scroll, this 'page mode' will be retained. Scrolling via the mouse-wheel, or via the arrow keys, will suspend PAGE mode till the next time an **UP PAGE** or **DOWN PAGE** command is used.

Handling files with lone CR characters

A “**lone CR**” character – that is, a CR not followed by LF, FF or another CR – is sometimes produced by older software that attempted to *overprint* the data in order to simulate underscores or bold print. It may also exist in non-Windows text files; older versions of Macintosh and some lesser-known systems used CR as a line termination. Because of this, a lone CR character might be used for two different, conflicting purposes. To handle this, you may choose between **EOL AUTO** and **EOL AUTONL**. These two options work as follows:

For all but the “lone CR” situation, **EOL AUTO** and **EOL AUTONL** work identically.

When **EOL** is set to **AUTONL** and SPFLite detects a lone CR in a file, it is considered to be a “new line” and is treated as if the lone CR were actually a normal CR/LF line termination.

When **EOL** is set to **AUTO** and SPFLite detects a lone CR in a file, it is considered to be an overprint request. At this point, SPFLite will buffer the lines involved in the overprint request until a ‘normal’ line terminator is found, and then it attempts to simulate an overprint. This means that, on a column-by-column basis, it examines the characters that are attempting to ‘occupy’ the same column at the same time. For each two characters involved in this way:

- When a blank and a non-blank character are in the same column, the non-blank character ‘wins out’.
- When two non-blank characters are in the same column, and they are identical, it is recognized as a “bold font” type of overprint, and is not a problem; the non-blank character is retained.
- When two non-blank characters are in the same column, but one of the non-blank characters is an underscore, the non-underscore character ‘wins out’. That way, the underscore character will never ‘obliterate’ the meaningful data.
- When two non-blank characters are in the same column, and they are not identical, and neither is an underscore, an “overprint clash” has occurred. When this happens, the first such character is retained and any others are discarded. Where SPFLite detects this, it will issue a warning message. If you frequently see this warning, it suggests the file having lone CR characters was not written to overprint, but either has foreign line delimiters or is perhaps ‘damaged’ in some way. The way to address this is to change the setting from **EOL AUTO** to **EOL AUTONL**.

Working with Unicode Mapping Files

Contents of Article

[Introduction](#)

[How the Unicode Mapping File is defined](#)

[How the Unicode Mapping File works](#)

[A sample Unicode Mapping File application](#)

[Future plans for Unicode Mapping Files](#)

Introduction

If you have used one of the "Raster" fonts supplied with SPFLite, you may have noticed several "Ansi" characters that were not really Ansi at all. For instance, in location hex **8D** you will see a less-than-or-equal sign. That is **not** Ansi, but it **is** the same as the Unicode character **U+2264**.

As you may know, SPFLite does not directly operate with Unicode data, but only with Ansi. Internally, all data that SPFLite deals with is Ansi data. So how is it that a less-than-or-equal sign can appear in this position?

The reason is that the supplied Raster fonts were custom-designed for SPFLite, and the extra characters you see occupy "unofficial" (unassigned) positions. SPFLite does not know or care what a byte of hex **8D** means. It simply displays it, and allows the font definition to control what visually appears on your screen.

All that is fine, if the only thing you do is **display** your data. But what if you want to print it? Previously, SPFLite would simply send the Ansi data to your printer. When Windows receives this data, it internally converts the data as needed to Unicode. For example, in the Microsoft 1252 code page used for Ansi, the Euro symbol € appears at location hex **80**, while its Unicode value is **U+20AC**. If you have a Euro symbol in the data you print, the hex **80** value will eventually get converted to hex **20AC** so that your printer will recognize it, and the right character is printed. Windows can do this because it "knows" what Ansi characters mean in terms of their Unicode definitions, which are standardized and well-known.

The case where a character of hex **8D** is used is more complicated. That is because there **is no** mapping for hex **8D**. There isn't any, because we made it up. **It is a "private definition" that exists only within SPFLite**. We added it because people often want to use characters like , and in their data, but plain Ansi doesn't provide them. In the past, SPFLite would have transmitted a hex **8D** to your printer - but the printer would not know what to do with it, and it would either print it as a blank or as nothing at all (causing the line it was on to appear shifted over), or the printer might have printed some 'garbage' character instead.

The problem with printing hex **8D** as a less-than-or-equal sign is **not** that this character is "unknown" or "illegal". It's just that it is not known as **8D**. Somehow, we need to be able to convey to the printer that when an **8D** is encountered, we really want to print **U+2264**.

By providing a Unicode Mapping File, these unusual characters can be printed just like any regular data.

How the Unicode Mapping File is defined

A Unicode Mapping File is an ordinary text file that is located in the same **SPFLite** directory where your **SPFLite.INI** file exists. You will have to prepare and save this file into that directory yourself. A sample file appears below.

The mapping file must have one of the following names:

SPFLite.Unicode

SPFLite.Print.Unicode

For now, you can just use the short name, **SPFLite.Unicode**. See [Future plans for Unicode Mapping Files](#) for more information.

If neither of these files exist in the SPFLite directory, SPFLite will handle printing the same as previously, with no character mapping performed. It is not an error if one of these files does not exist, and no message will appear.

A Unicode Mapping file contains the following elements:

- A "title" line containing **[Unicode]**
- Optional blank lines, which are ignored
- Optional comments, starting with a leading ; semicolon, which are ignored
- One or more mapping lines.

A *mapping line* has the following format:

Xnn=Unnnn

The left side with **Xnn** defines the Ansi character you wish to map. Let's say we want to map the less-than-or-equal character. So, the **Xnn** would appear as **X8D**. The right side with **Unnnn** defines what Unicode character you want to print. The Unicode value for this character is **U2264**.

The + plus sign, used to describe Unicode in formal publications, is not used here. Just say **U2264**, not **U+2264**.

You must specify exactly two hex digits for the **Xnn** side, and exactly four digits for the **Unnnn** side.

On a mapping line, blank characters are ignored. If you wish, you can place an optional comment on a mapping line, by following the **Unnnn** part with a ; semicolon and your comment. Here is an example of the less-than-or-equal mapping line, with a comment:

X8D=U2264; less than or equal

In case the same **Xnn** value appeared more than once in the same Unicode Mapping File, the final one will override any prior entries. You can use this fact to store alternative mappings in your file. You would place the mappings you really want as last in your file, and prior ones would essentially be treated as comments. That way you can keep the unused, alternative values in your file without deleting them.

If any format errors are found in your Unicode Mapping File, an error message is displayed, and SPFLite will not use your mapping file, but will print your data file the same as previously, without any character mapping performed.

How the Unicode Mapping File works

When you request SPFLite to send data to the printer, it looks for a Unicode mapping file in the **SPFLite** directory, where the **SPFLite.INI** file is located. It will first look for a file called **SPFLite.Print.Unicode**, and will use it if one is there; otherwise it will look for a file called **SPFLite.Unicode**.

When a Unicode Mapping File is found, the entries are validated and stored into memory. From that point on, any print requests will be transmitted to your printer directly in Unicode, rather than allowing Windows to convert your data to Unicode itself.

The trick here is that by defining your own Unicode Mapping File, **you** decide which specific Unicode value is sent for any given Ansi character.

It is not necessary to map every Ansi character to Unicode. Only map the characters that you really need to change from the usual definitions. For instance, the digit 5 is **35** in Ansi and **U+0035** in Unicode. There is no reason to map such a character - unless you're doing something very unusual. So, even though a Unicode Mapping File could theoretically have 256 unique mappings, you are not likely to ever actually specify that many.

While these unusual characters will likely only appear on your screen in the fonts provided with SPFLite (the Raster screen fonts), you can use **any** True Type font to print with, provided it has the necessary characters. Because real True Type data is being sent to your printer, you are not limited to a printer font like **RasterTTF** or our new **RasterTN** True Type font. For instance, a Microsoft fixed font called **Consolas** may be used to print these characters. You can also use **Courier New**, although you will need to confirm that this font has all the characters you might need. There may be many other font choices available to you.

You can use the Windows utility **charmap** to see what characters exist in a given font, or you could use a program like Microsoft Word, and select **Insert > Symbol** to see what is available. These tools will also show you the Unicode hex value you will need for your Unicode Mapping File.

Asample Unicode Mapping File application

Here is a display of our revised Raster15 TN screen font. It is similar to the prior Raster15 font, but the extra characters you see are designed to make all of the symbols on an IBM 1403 printer with a TN print chain available. The particular placement of characters is dictated by the available free positions, and some values don't work well as printable data, like **CR**,**LF** and **ESC**.

You may also notice that the added characters don't include superscript 1, 2 and 3, because they already existed in Ansi. You now have the full set of superscripts from **0** to **9**, just not in adjacent locations.

These new screen fonts, and the new **RasterTN** True Type font, will be made available on the SPFLite web site about the same time that the Unicode Mapping File support is released. Check the web site for more information.

053 X'35' 5 added to Clipboard															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
10	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
20	!	"	#	\$	%	&	'	()	*	+	,	,	,	,
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	~
80	€	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ	ƒ
90	■	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À
D0	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð	Ð
E0	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à
F0	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ	đ

You will notice that the "unusual" characters are in the Ansi range of hex **00** to hex **1F**, and a few in the range of hex **80** to **9F**.

Here is a sample **SPFLite.Unicode** mapping file that will allow you to print this data.

[Unicode]

```
X00=U0020; null as space
X01=U2502; box vertical
X02=U2534; box T up
X03=U252C; box T down
X04=U2191; arrow up
X05=U2193; arrow down
X06=U2192; arrow right
X07=U2190; arrow left, note that ASCII 07=BEL
X08=U0020; control BS as space
X09=U0020; control HT as space
X0A=U0020; control LF as space
X0B=U0020; control VT as space
X0C=U0020; control FF as space
X0D=U0020; control CR as space
X0E=U251C; box T right
X0F=U2524; box T left
X10=U2070; superscript 0
X11=U207A; superscript +
X12=U207D; superscript (
X13=U207E; superscript )
X14=U2074; superscript 4
X15=U2075; superscript 5
X16=U2076; superscript 6
X17=U2077; superscript 7
X18=U2078; superscript 8
X19=U2079; superscript 9
X1A=U0020; control EOF as space
X1B=U0020; control ESC as space
X1C=U2514; box LL
X1D=U2518; box LR
X1E=U250C; box UL
X1F=U2510; box UR
X7F=U0020; control DEL as space
X97=U2500; box horizontal
X81=U253C; box intersection
X90=U25AA; black square
X8D=U2264; LE
X8F=U2260; NE
X9D=U2265; GE
XA0=U0020; non-breaking space
XAF=U207B; superscript - 00AF may be OK too
```

Notice that if you had data with embedded **CR** and **LF** characters (say, if your file was **RECFM F** and **EOL NONE**), that might be valid as screen data while editing, but your printer may object. In addition, our Raster screen fonts have the special **CR** and **LF** symbols as single characters, but most printer fonts do not have these defined. To avoid such data causing problems for your printer, you should map these control codes to space (**U0020**) so that it won't interfere with the printer's normal operation.

If you have a regular text file, rather than one with **EOL NONE**, SPFLite takes care of the line termination for you. Don't worry - even though you might have **CR** and **LF** mapped to U0020 as shown above, this **won't** change how a regular text file is printed. You'd only get **CR** and **LF** changed to blanks if they were **data** values in your file (in an **EOL NONE** file), but not when they are part of ordinary text file lines.

Future plans for Unicode Mapping Files

In theory, based on the way we have designed the naming conventions for Unicode Mapping Files, it is conceivable that some day we could add support for mapping of screen fonts, in addition to printer fonts. If this were done, you could have one common mapping file for both, using a file of **SPFLite.Unicode**, or if the mappings were different, you could have an **SPFLite.Print.Unicode** and an **SPFLite.Display.Unicode**. That would make it possible to use any fixed font as a display font (rather than just our special Raster screen fonts), and set aside certain special characters as alternative Ansi codes.

In theory, we could also tailor the Unicode mapping of files differently, depending on their file type, by adding some kind of enhanced **PROFILE** support. If that were done, the default prefix of **SPFLite** on a **.Unicode** file would be changed to something else.

Such changes would be more involved than the basic Unicode mapping support we have presently added for printers. We are not making any promises as to if or when support for these features might be added. Any enhancements that are made will be announced on the SPFLite forum if and when they become available.

Managing Line Lengths

Contents of Article

[Introduction](#)

[The MINLEN profile option](#)

[The TR and PL line commands](#)

Introduction

SPFLite gives you the control over the lengths of lines of text files. This is done primarily by the command [MINLEN - Set Minimum Record Length](#).

Why would this issue be important to you? Here are some likely reasons:

- You are using Picture strings in conjunction with primary commands like **FIND**, **CHANGE**, **EXCLUDE**, **SPLIT** and **JOIN**, and sometimes run into issues with "blank" lines that are zero-length. If you could avoid creating zero-length lines, some types of editing tasks can be done more simply.
- You may be creating a conventional text file with standard **EOL** delimiters, but for some reason you need all lines to be of the same length. This requirement may stem from usage of the file by an external system such as a mainframe, or perhaps by an embedded system.

The MINLEN profile option

To define a "garden variety" standard, variable-length Windows text file, the associated PROFILE settings are **RECFM U**, **LRECL 0**, **EOL CRLF**. The **LRECL** option is named after an IBM term meaning Logical Record Length. In SPFLite, this term really means the **maximum** logical record length. For files of type **RECFM F** and **RECFM V**, the **LRECL** must be a positive number defining the maximum allowed line size. For **RECFM U**, **LRECL 0** really means that there **is no** (arbitrary) **maximum length**, because the end of the line is determined from the **EOL** setting rather than a fixed value.

In addition to the **LRECL** option, you can now define the **minimum** logical record length using **MINLEN**.

In prior versions of SPFLite, the minimum logical record length is treated as **0**; that is, there **is no** minimum, so text lines can be of any length, including length zero.

The **MINLEN** option defaults to **0** unless you set it otherwise.

When the **MINLEN** option is set, SPFLite does the following:

- If the **MINLEN** value is greater than zero, the file currently being edited is scanned, any any lines shorter than the newly-specified **MINLEN** are padded with blanks to the minimum length.
- Any lines changed manually by typing into them, lines changed from primary or line commands, and lines added from **COPY** and **PASTE** commands, will have their minimum line length enforced by the **MINLEN** value in effect at the time.

The **MINLEN** option, being a PROFILE setting, is unique to a given file type. If you want a certain minimum length enforce (like 1 for example), you would have to update all of your existing profiles, and then update the **DEFAULT** profile so that any newly-created profiles would also have the **MINLEN** set to what you wanted.

Note: Be aware that any primary command such as **SPLIT** and **JOIN**, and line commands such as **TS**, **TB/TBB**, **TR**, **PL** and **TL** may be documented as implicitly producing lines of a certain length, but the **MINLEN** value in effect at the time for a file will **override** this, so that no matter what you attempt to do, you will not create lines shorter than the minimum length.

The TR and PL line commands

If you need to truncate or pad lines to manually enforce a line length, the Trim command **TR/TRR** and the Pad to Length command **PL/PLL** can be used.

The Pad to Length line command **PL** will accept a special modifier of `/` or `\`. When used in this way, a command of **PL/** will pad all following lines to a minimum length of 1; **PL** will do the same to preceding lines. Placing **PL/** on line 1 of a file can be used to ensure that all lines in the file have a minimum line length of 1; that is, it is a quick way to ensure there are no zero-length lines in the file. If you need to pad to a longer length (such as 4 for example), you can use the block mode version with **PLL4** on line 1 and **PLL** on the last line.

The Truncate line command **TL/TLL** can also be used, but be aware that since truncation is involved, this command can cause data loss.

See [TR / TRR - Trim Trailing Blanks](#), [PL / PLL - Pad Lines](#), and [TL / TLL - Truncate Lines to a Length](#) for more information.

Working with Read Only Files

Contents of Article

[Introduction](#)

[Determining that a file has the Read Only attribute](#)

[Accessing Read Only files from the SPFLite File Manager](#)

[Editing Read Only files from an SPFLite Edit or Browse window](#)

[Read Only files and MEDIT](#)

[Actions taken during END or SAVE](#)

[Actions taken during Rename or Delete](#)

[Actions requiring Windows Explorer or other external intervention](#)

Introduction

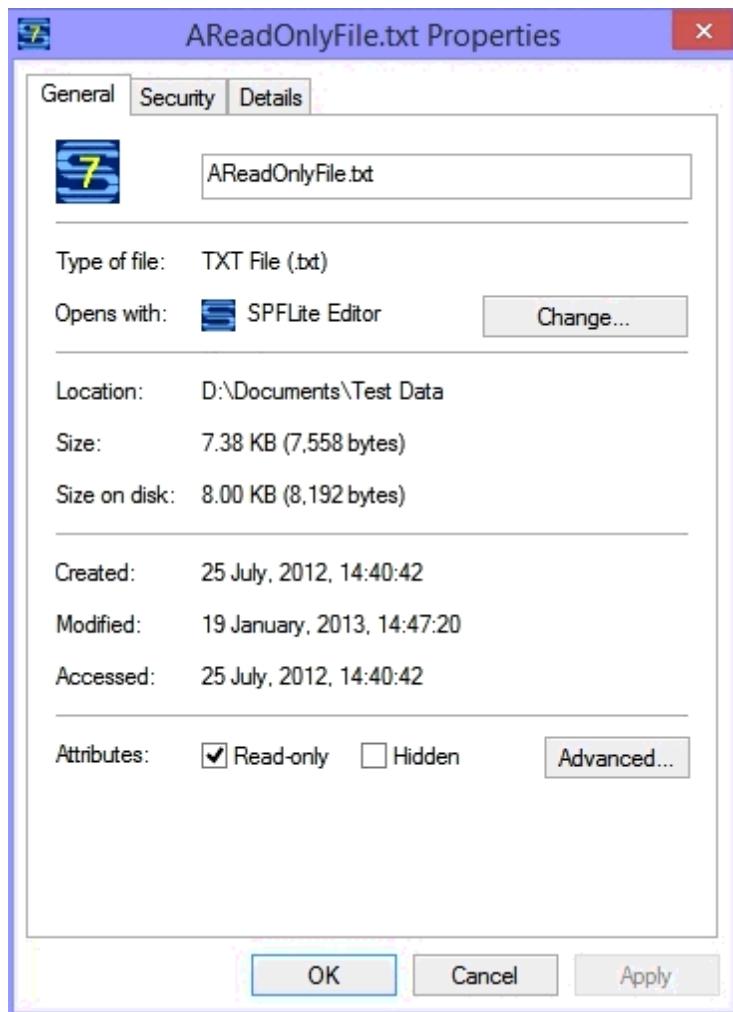
SPFLite will respect the Read Only attribute of a file. Because this action takes place automatically, there is not a lot for you to do, except to understand the process SPFLite uses to handle Read Only files.

Determining that a file has the Read Only attribute

[From the Windows Explorer](#)

When Explorer displays a list of files, you can see the Read Only attribute (among other things) if you add the **Attributes** column heading to the display. To do that, right-click on the Explorer heading line on some empty area, and a box of heading types will appear; click on **Attributes**. This column does not typically appear by default, unless you have made a global change to Explorer to have every file display contain this heading.

To see the properties dialog for a particular file, where you can both display and change the Read Only attributes, right-click on the file name, and select Properties. You should see various attribute check-boxes, including Read Only. Here is a sample Windows Explorer display:



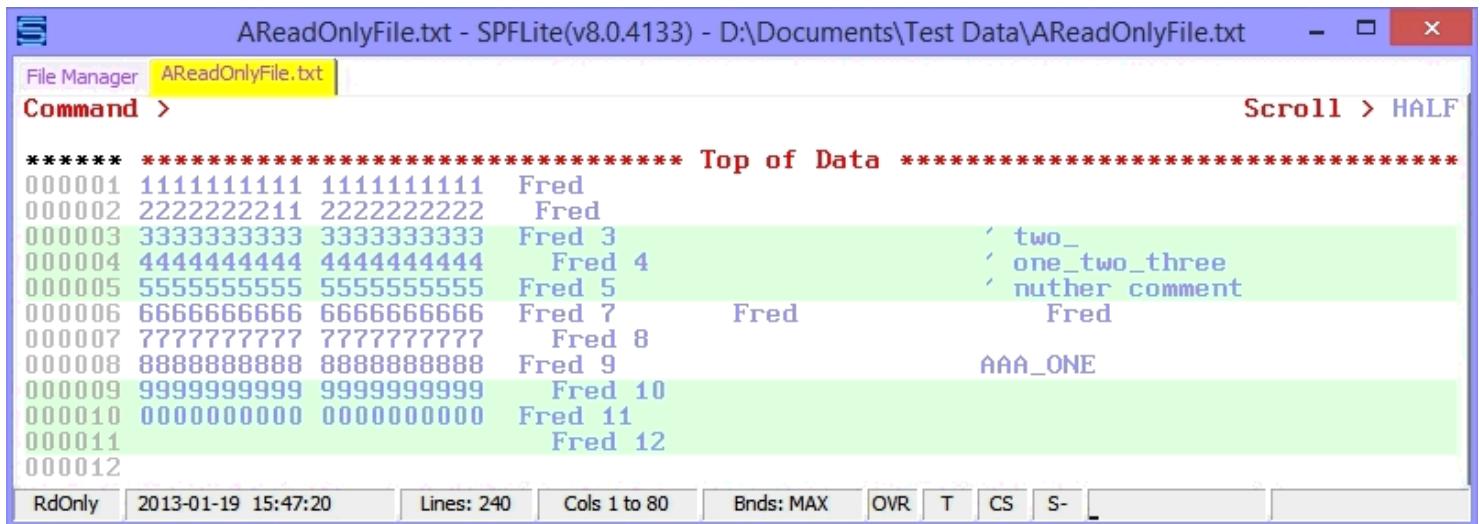
Accessing Read Only files from the SPFLite File Manager

When a file contains a Read Only attribute, the File Manager will change the line-command field for that file, so that it displays as **dots** instead of underscores. This is a "flag" to remind you that you are dealing with a Read Only file. Here is a sample File Manager display:

Cmd	Name	Size	Date
	AlternateBackground.txt	5.5K	2013-05-05 13:40
	ANoExtensionFile	240	2013-01-19 16:25
.....	AReadOnlyFile.txt	7.4K	2013-01-19 15:47
	BadCUE.txt	545	2013-03-25 14:22
	blanks.txt	63	2012-02-29 13:53
	BOUNDS-PROBLEM.TXT	723	2014-03-24 13:32

If you click on a file name, or use the File Manager line commands **E** (Edit) or **S** (Select), SPFLite normally will open an **Edit** session for the selected file. When the file has a Read Only attribute, the file will be opened in a

View session rather than an **Edit** session. SPFLite detects the Read Only attribute and automatically switches the action from Edit to View. The first box of the Status Bar will, instead of simply showing **View** will say **RdOnly**. Technically, this is only a cosmetic matter. You are actually still in **View** mode. The reason for changing the indicator is so you won't forget **why** you are in Browse mode - it's because the file was found to be a Read Only file.



The screenshot shows the SPFLite interface with the file 'AReadOnlyFile.txt' open. The status bar at the bottom indicates 'RdOnly'. The data is displayed in a grid format with columns. A tree view on the right side shows a structure with 'two', 'one_two_three', and 'nuther comment' as children of the first row. The status bar also shows the date and time (2013-01-19 15:47:20), line count (Lines: 240), column range (Cols 1 to 80), and various mode indicators (Bnds: MAX, OVR, T, CS, S-).

***** * Top of Data *****				
000001	1111111111	1111111111	Fred	
000002	2222222211	2222222222	Fred	
000003	3333333333	3333333333	Fred 3	' two_
000004	4444444444	4444444444	Fred 4	' one_two_three
000005	5555555555	5555555555	Fred 5	' nuther comment
000006	6666666666	6666666666	Fred 7	Fred
000007	7777777777	7777777777	Fred 8	
000008	8888888888	8888888888	Fred 9	AAA_ONE
000009	9999999999	9999999999	Fred 10	
000010	0000000000	0000000000	Fred 11	
000011			Fred 12	
000012				

Editing Read Only files from an SPFLite Edit or Browse window

If you issue the **EDIT** primary command from an existing Edit or Browse session, SPFLite detects the Read Only attribute and automatically switches the action from Edit to View. You will get the same message here as you would when trying to edit a file from the File Manager.

Read Only files and MEDIT

SPFLite does not currently support Read Only files participating in Multi-Edit sessions. If you want to use a Read Only file within **MEDIT**, you must remove the Read Only attribute, or copy the file to another name, and ensure that the copy of the file is not Read Only.

Actions taken during END or SAVE

If you attempt to **SAVE** a Read Only file, the command will be rejected with an error message. If you have made changes to the file, these changes will not be saved during **END**. Instead you will see a popup message asking you to continue the **END** operation without saving. **Yes** is the same as a **CANCEL** command, and **No** causes the **END** to be ignored and you are returned to the Read Only browse screen. This is done regardless of the AUTOSAVE options currently in effect.

Actions taken during Rename or Delete

SPFLite will not allow you to rename or delete a Read Only file, and will issue an error message informing you of that. This applies to the Delete and Rename line commands (**D** and **R**) in File Manager, as well as the **RENAME** and **CANCEL DELETE** or **CANCEL PURGE** primary edit command.

Actions requiring Windows Explorer or other external intervention

During the design phase for respecting the Read Only attribute of files, consideration was given for allowing Delete and Rename capabilities, perhaps with a warning popup message, as well as the ability to set and clear the Read Only attribute from File Manager. However, if it's too easy to circumvent the protection, it can be almost as bad as not having any protection at all. As a result, the decision was made not to allow this.

Because of that, in order to provide the most protection for these files, if you have to take these actions, you will have to do so outside of SPFLite, and use the Windows Explorer or a command line prompt to rename or delete

a Read Only file, or modify the Read Only attribute of a file.

Handling External File Changes

Contents of Article

[*Introduction*](#)

[*Using the Global Options dialog*](#)

[*Using the NOTIFY command*](#)

Introduction

SPFLite can be directed to inform you when a file you are working on in an Edit or Browse session has been modified by some process outside of SPFLite itself.

How could this happen? Well, suppose you are using SPFLite to edit a source program, but you are also using the IDE of a compiler to test your program, and that IDE also has its own editor. If you make a change in the source program within the IDE while SPFLite has the same file open, it appears to SPFLite as an external modification.

You are provided with a finer control over the conditions under which you will receive a notification. That means that you can decide how important it is to be informed that such a change has happened.

You have two means by which to manage external file modifications. One is a permanent file notification setting through the Global Options dialog, and the other is a temporary file notification setting by using the **NOTIFY** command. These settings always apply to all files of all types. You cannot control notifications for a file type or a specific file.

Using the Global Options dialog

The permanent file notification level is defined in the [General tab](#) of the Global Options dialog with the [Notify tabs on external file change](#) dropdown box, and is either **ALL**, **NONE** or **EDIT**.

As a global option, this setting is permanent across SPFLite invocations, and is saved in the SPFLite.INI configuration file. It affects files of all file types, and is not dependent on a file's PROFILE settings. These options have the following meaning:

ALL

SPFLite will notify you in all cases when you have a file opened for Edit or Browse, and the file has been modified from outside. When this happens, you will be given an opportunity via a popup, to reload the file at that time. If you would rather reload the file later (or not at all), you can click on Cancel, and then issue the **RELOAD** primary command later if you wish.

NONE

SPFLite will **not** notify you in when you have any file opened for Edit or Browse, and the file has been modified from outside. The reason you might choose the **NONE** option is that you may be well aware of the reasons why your files are getting modified from the outside (probably because **you** are the cause of it) and you don't want to be bothered with receiving and replying to these notifications.

EDIT

SPFLite will notify you in all cases when you have a file opened for Edit (**but not for Browse**), and the file has been modified from outside. The reason you might choose the **EDIT** option is that, if you have a file open for Browse, you may not care if a file has been updated from outside, since you are not modifying it yourself within SPFLite anyway, and will not be saving it. It would be more important to be made aware of this situation in an Edit session, to avoid overwriting a file modification originating elsewhere.

Using the NOTIFY command

The **NOTIFY** command is used to set the temporary notification level for files opened in SPFLite that are modified by an external process. The setting you choose on **NOTIFY** only lasts until the next **NOTIFY** command, or until SPFLite is terminated.

When **NOTIFY** is used with the **ALL**, **NONE** or **EDIT** option, it sets the notification level as described above. The only difference between using the **NOTIFY** command and setting the option in the Global Options is that the notification level is not permanently saved.

A **NOTIFY RESET** command will restore the notification level to that defined in the Global Options.

A simple **NOTIFY** with no operand will report the current notification level in effect as of that time.

See [Options - General](#) and [NOTIFY - Set Temporary File Notification Level](#) for more information.

Working with the LINE Primary Command

Contents of Article

[Introduction](#)
[Dual-use and comparable commands](#)
[Fundamentals of the LINE primary command](#)
[Applications of the LINE primary command](#)
[Specifying the line-command operand](#)
[Specifying a block-mode line-command operand](#)
[LINE provides access to excluded lines](#)
[Using Labels and Tags with the LINE Command](#)
[Limitations of T/TT line command and LINE primary command](#)
[LINE primary command examples](#)

Introduction

The **LINE** primary command allows you to apply a line command to one or more lines, based on a standard line-range operand. The command syntax is described in [LINE - Apply Line Command](#).

This is a somewhat unusual concept, not present in ISPF or in SPFLite until now, and is best explained by example - so we provide several to help you to understand it and get most out of this new feature.

You can go directly to the examples [here](#).

Note: Every effort was made to carefully design and implement it, but because this a novel editing concept (even for us, it must be admitted), the way in which this works may be subject to change if it turns out any of our basic assumptions need to be adjusted. We invite all SPFLite users to try this and give us your suggestions and feedback on how well you do or don't like it. This a very cool and promising feature, but it's also a work in progress, so stay with us as we go through the learning curve together.

For you mathematicians out there, note that the **LINE** primary command is a function that applies other functions (line-commands) to your data, which could be termed a lambda function. For everyone else, it's just a cool feature, so keep reading ...

By the way, since **LINE** is a primary command, and commands that go into the sequence area of lines are "line commands", the terminology could get a little confusing if we're not careful. To avoid that, we will always refer to **LINE** as "**the LINE primary command**" or "**the LINE facility**" and continue to call commands in the sequence area simply as "**line commands**". What we **won't** do is refer to "**the LINE command**".

Dual-use and comparable commands

Even though the **LINE** primary command is new, if you are a long-time SPFLite user, you may already have taken advantage of a similar capability. Consider the **LC** command, to convert data to lower case. This command exists in two forms, as a line-mode **LC[n]** or block-mode **LCC** line command, and as a primary command having a syntax of:

LC [line-control-range] [X | NX | ALL] [MX | DX]

Because you have the ability to use **LC** in both line-command form and primary-command form, **LC** could be thought of as a "dual-use" command. It is **as if** you had a "primary-command version of a line-command", or vice-versa.

There are only a limited number of commands that are dual-use. (A few of these might be termed "comparable"

dual-use commands, like the line command **D** and the primary command **DELETE**. Both have the comparable function of deleting lines, but they are spelled differently.)

These dual-use commands are:

Line command	Primary command	Note
D / DD	DELETE	Comparable
LC / LCC	LC	
S / SS	SHOW	Comparable
SC / SCC	SC	
TC / TCC	TC	
UC / UCC	UC	
X	EXCLUDE	Comparable. EXCLUDE can also be abbreviated as X, the same name as the line- command form

Even though it might **seem** like the **LC** line command and the **LC** primary command are "the same" (because both perform lower-case conversion), they are in fact completely different commands in different parts of SPFLite and were implemented independently. They just happen to be spelled the same.

Fundamentals of the **LINE** primary command

The **LINE** facility is different. In effect, the **LINE** primary command is used to "deposit" line commands into the sequence areas of one or more data lines, at which point those line commands are **applied**, as if you had typed them in yourself and pressed Enter. When that line command is executed, it's **not** a "primary-command version of a line-command". It's a **real** line command, and it works like the real thing.

That is, suppose you have a line labeled **.ABC** and you want to convert that line to lower case. You have three choices:

- Type an **LC** line command on the line labeled **.ABC** and press Enter.
- On the primary command line, type the command **LC .ABC** and press Enter.
- Use the **LINE** primary command to apply an **LC** line command to the line. Referring to the example above, this would be done with the primary command **LINE LC .ABC**.

The difference between **LC .ABC** and **LINE LC .ABC** is:

- **LC .ABC** directs all of its activity from the primary command line, whereas,
- **LINE LC .ABC** is a request to apply the **LC** line command on line **.ABC**. When **LINE LC .ABC** is executed, a real **LC** line command is executed.

Note: Just to be clear, this discussion about **LC** is simply to help you understand the process. Even though you could do it, you wouldn't normally need to say **LINE LC**, since you already have an **LC** primary command available.

However, if you were to use any of the extra operands of **LINE**, like **FIRST**, **LAST**, **X|NX**, and/or a multi-line line range using labels or tags, there are some reasons to consider using a command like **LINE LC**.

Applications of the **LINE** primary command

If you read no further, you might conclude that the **LINE** primary command wasn't all that useful. After all, if it acts just like typing a regular line command, why not just type a **real** one and be done with it? What is the advantage to using the **LINE** facility? It was created to address the following situations:

- As a way to apply a given line command to a range of lines, including multiple, non-contiguous lines specified by line label ranges, line tags and/or by their **X|NX** exclusion status. Applying the **LINE** facility to multiple data lines is its most powerful feature, and is likely to be the most important use for the **LINE** primary command.
- As a means to apply line-commands to a file from a command macro, which otherwise would have no way to do this.
- To provide additional capability to keyboard macros.
- And, to provide a critical feature needed for the SPFLite programmable macro facility.

Specifying the line-command operand

The **line-command** operand may be a quoted or unquoted string.

The operands of the **LINE** primary command may be specified in any order. We show the **line-command** operand in the examples as immediately following the **LINE** primary command keyword just as a matter of style and convention.

When the **line-command** operand is a simple alphabetic or alphanumeric string, like **R** or **R2**, it is not necessary to quote it. Either of these will work:

```
LINE R .123
LINE 'R' .123
```

You can (surprisingly) even use the "graphic" shift commands without quoting them. Either of these will work:

```
LINE )4 .123
LINE ')4' .123
```

However, if you try to use a line-command operand that overlaps the remaining syntax of the **LINE** primary command, you will run into problems. The problem areas are as follows:

- A line-command operand of **X** will be confused with the **X|NX** option of the **LINE** primary command. If you try something like **LINE X .123** because you wanted to put an **X** on line **123**, you will get an error message, "**Missing/invalid line command**". To get around this, you will have to quote the line-command operand as **LINE 'X' .123**.
- If you attempt to use the **LINE** primary command to place a **label** into a line, it will be confused with a label that is part of the line-range operand. If you try something like **LINE .ABC .123** because you wanted to put a label **.ABC** on line **123**, you will again get an error message, "**Missing/invalid line command**". To get around this, you will have to quote the line-command operand as **LINE '.ABC' .123**.
- If you attempt to use the **LINE** primary command to place a **tag** into a line, it will be confused with a tag that is part of the line-range operand. If you try something like **LINE :T .123** because you wanted to put a tag **:T** on line **123**, you will get an error message, "**Invalid tag operand**". To get around this, you will have to quote the line-command operand as **LINE ':T' .123**.

If you don't want to think about it, you can just quote all **line-commands**. But, remember to at least quote the **X** if it's a **line-command operand**. If you are using and **X** to select just excluded lines as part of the line-range, then

don't quote it.

Note: The **LINE** primary command operands **FIRST**, **LAST**, **ALL** and **NX** should **never** be quoted, because these cannot be **line-commands**.

Note: Would you ever have an 'X' in quotes **and** an unquoted **X** in the same **LINE** primary command? You could, but, there's no need for it, because you would be putting **X** line-commands on lines that were already excluded. While a command like **LINE 'X' X .123 .456** is not useful, you **could** use **X** with a line count, like **LINE 'X3' X .123 .456**.

You can also use the block mode **XX** line command with **LINE** primary command. The main thing is, if it makes sense to do it manually, it should make sense to do it with the **LINE** primary command.

If the **line-command** operand is a quoted string of a blank, it will erase a label or tag that exists on that line. (You can't use a ' ' zero-length string to do this; there must be at least one blank character in the string.) For example,

```
LINE '.ABC' .123
```

will place label **.ABC** on line 123, and

```
LINE ' ' .123
```

will remove that label. You may use the forms

```
LINE '..ABC' .123
LINE '::DEF' .123
```

to 'toggle' a line label or line tag on a line, and you may use the forms

```
LINE '..' .123
LINE '::' .123
```

to specifically remove either a line label or line tag on a line, in cases where both coexist on the same line. In other words, it works just like you could do with a keyboard macro.

When you apply the **LINE** facility to a line that already has a label or tag, the label or tag will not be disturbed. But if you have a pending command, the label or tag will be 'concealed' until the operation is completed, and then it will be redisplayed. This is the same as what happens manually.

Specifying a block-mode line-command operand

You are allowed to use the **LINE** primary command to place a block-mode **line-command** into a line. For example, the following is valid:

```
LINE CC .123
```

What happens when you do this? Basically, the same thing that would happen if you did this manually. You will see the **CC** line command sitting in the sequence area of line 123 when this is performed.

Depending on whether there is another **CC** and/or **A/B** command somewhere in the file, SPFLite will report the situation with a message like **Pending command(s)**, or it will perform the command and then clear any messages if "all the pieces are in place". If you tried to put too many block-mode **line-commands** into the file, such as with a command like,

```
LINE CC .1 .10 ALL
```

you will get a message, **Illogical line command grouping**, because it wouldn't make sense to have ten pending **CC** commands at the same time. The point is, SPFLite will do the same thing it would if you were typing everything in manually.

LINE provides access to excluded lines

Normally, when a block of lines are excluded, you are limited as to what you can do to those lines using line commands. You can do things like delete, repeat, shift, and copy, using a line-mode command on the excluded-line placeholder, but these actions are applied to **every** line in the excluded region. You can't pick and choose which lines **within** that region are affected.

With the **LINE** primary command, this changes. Suppose you have lines 1 to 100 excluded, and you want to delete line 40, then repeat line 20, but you **also** want the rest of the lines to remain excluded. How would you do it? The **LINE** primary command provides you line-command access to those excluded lines. Here's an example of how you could do it.

```
EXCLUDE .1 .100 ALL
LINE D .40
LINE R .20
```

At this point, you will still have 100 excluded lines, and you have modified some of the data within the excluded region without anything 'popping out'. Try this test yourself - then, unexclude the file with a **RESET** primary command and see what you have.

This kind of capability never existing in SPFLite before, and it opens up many intriguing possibilities. Feel free to experiment. If you happen upon some novel application of the **LINE** primary command, we invite your feedback on the SPFLite web site. If you have a good idea, we could include it in upcoming revisions to the Help document.

Using Labels and Tags with the LINE Command

You can manage line labels and line tags using the **LINE** primary command. For example, to put a label of **.ABC** on line 10, you can issue the command:

```
LINE '.ABC' .10
```

Remember to quote the label, because in this example, the first operand **'.ABC'** is what goes into the sequence area, and the second operand **.10** is where the first operand is placed.

When doing a normal edit, there is usually no need to do such a thing; you would just put the label on the line directly. However, when run a programmable macro the **LINE** primary command is the only way to do this. You would need to 'wrap' the **LINE** primary command in an **SPF_CMD** function call in your **.MACRO** file, like this:

```
SPF_CMD ("LINE '.ABC' .10")
```

It is possible to set, clear and toggle both line labels and line tags from the **LINE** primary command. The possible options are as follows:

LINE '.label'	-- Enter label into sequence area
LINE '..label'	-- Toggle label; enter label if no label present, -- else clear any label that is present
LINE '..'	-- Clear any label that may be present
LINE '...'	-- Clear any label that may be present; same as LINE '..'

```

LINE ' :tag'                                -- Enter tag into sequence area

LINE ' ::tag'                                -- Toggle tag; enter tag if no tag present,
                                                -- else clear any tag that is present

LINE ' : '                                    -- Clear any tag that may be present

LINE ' :: '                                  -- Clear any tag that may be present; same as LINE ' : '

```

See [Special Line Commands](#) in [Using the KEYMAP Dialog](#) for more information on performing these same actions within a KEYMAP definition.

Limitations of T/TT line command and LINE primary command

The **T/TT** line command can be used by the **LINE** primary command, to apply **T/TT** to one or more lines. However, when **T** is applied to more than one line, each individual **T** is applied to each line one at a time. The way that **T/TT** operates, only a single, contiguous block of highlighted data may exist as any given time. So, if you attempted to issue a command like **LINE T .11 .13 ALL**, only line 13 will be highlighted. If you try to manually highlight line 11 with a **T**, then line 12, then line 13, you will see how and why it works this way.

LINE Command Examples

LINE Example 1: Insert blanks lines

LINE N .1 .3 is used to insert a blank line after the first 3 lines of the file.

```

File Manager Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT
Command > LINE N .1 .3
Scroll > HALF

***** **** Top of Data ****
000001 one
000002 two
000003 three
000004 FOUR
000005 FIVE
000006 SIX
***** **** Bottom of Data ****

Edit 2014-05-13 12:09:09 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+

```

Result:



File Manager Line-Cmd.TXT

Command > Scroll > HALF
'N' applied 3 times

```
***** **** Top of Data ****
000001 one
000002
000003 two
000004
000005 three
000006
000007 FOUR
000008 FIVE
000009 SIX
***** **** Bottom of Data ****
```

Edit * L 000002 C 1 Lines: 9 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0000

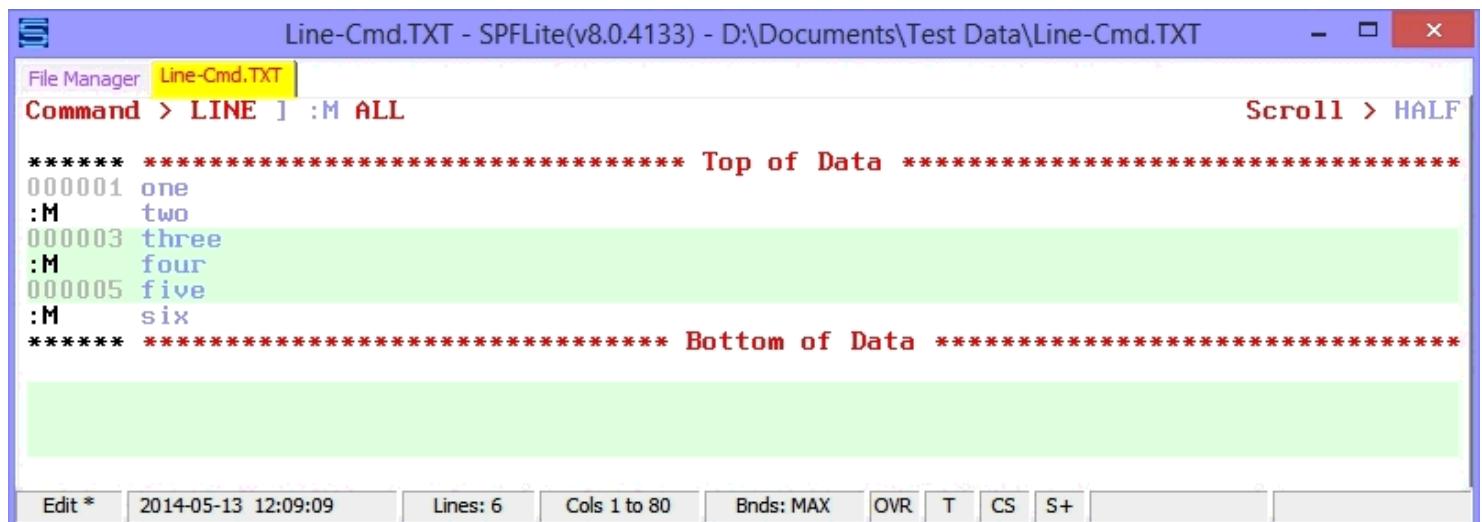
LINE Example 2: Repeating lines

LINE R .4 .6 is used to individually repeat 3 lines of the file. Note that this could **not** be done with a conventional **RR** block-mode line command.

Result:

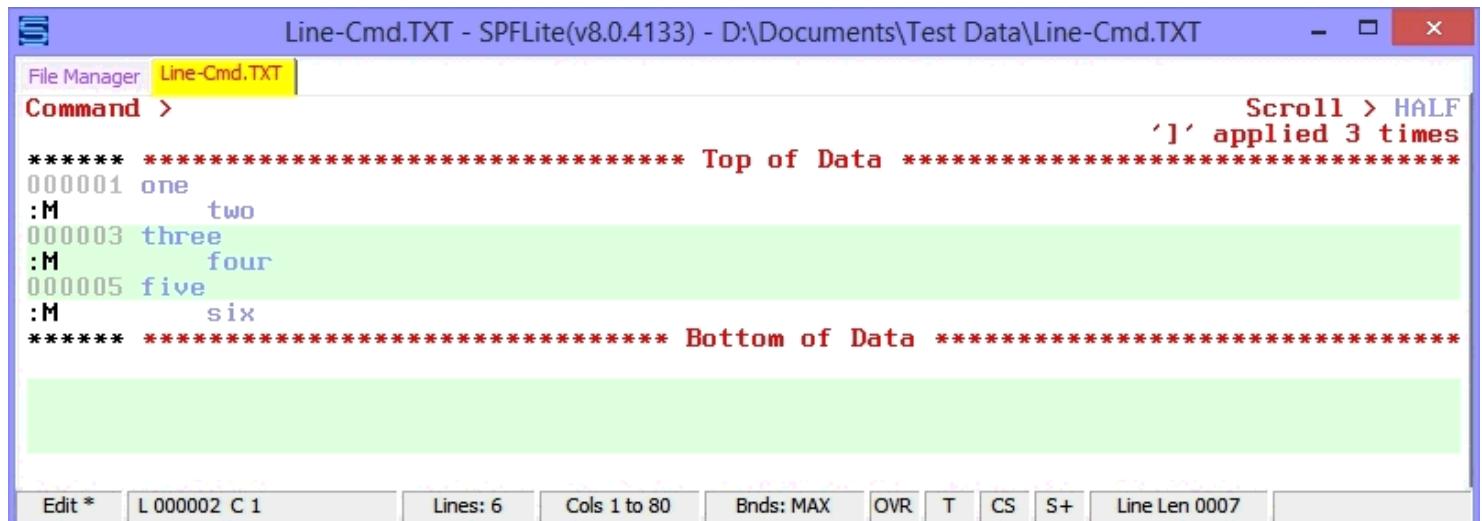
LINE Example 3: Shifting tagged lines

LINE] :M ALL is used to shift all lines with line tag :M on them.



```
File Manager Line-Cmd.TXT | Command > LINE ] :M ALL | Scroll > HALF
*****
000001 one
:M two
000003 three
:M four
000005 five
:M six
*****
Edit * 2014-05-13 12:09:09 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007
```

Result:



```
File Manager Line-Cmd.TXT | Command > | Scroll > HALF
'] applied 3 times
*****
000001 one
:M two
000003 three
:M four
000005 five
:M six
*****
Edit * L 000002 C 1 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007
```


Line Commands

Descriptions of the following Line Commands are contained in this section:

<u>A</u>	After destination
<u>AA</u>	After block
<u>B</u>	Before destination
<u>BB</u>	Before block
<u>BNDS</u>	Display BOUNDS line
<u>C</u>	Copy line(s)
<u>CC</u>	Copy a block
<u>COLS</u>	Display Columns line
<u>D</u>	Delete line(s)
<u>DD</u>	Delete a block
<u>E</u>	Display first lines in excluded region
<u>G</u>	Glue lines together
<u>GG</u>	Glue lines together in block
<u>H</u>	Here-destination lines
<u>HH</u>	Here-destination block
<u>I</u>	Insert temporary new line(s)
<u>J</u>	Join lines together
<u>JJ</u>	Join lines together in block
<u>L</u>	Display last lines in excluded region
<u>LC</u>	Lower-case lines
<u>LCC</u>	Lower-case lines in block
<u>M</u>	Move lines
<u>MM</u>	Move lines in block
<u>MARK</u>	Set column markers
<u>MASK</u>	Set the Insert line model
<u>MD</u>	Make data lines
<u>MN</u>	Make NOTE lines
<u>N</u>	Insert permanent new line(s)
<u>NOTE / xNOTE</u>	Insert NOTE lines
<u>O</u>	Overlay lines
<u>OO</u>	Overlay lines in block
<u>OR</u>	Overlay-Replace lines
<u>ORR</u>	Overlay-Replace lines in block
<u>PL</u>	Pad lines
<u>PLL</u>	Pad lines in block
<u>R</u>	Repeat lines
<u>RR</u>	Repeat lines in block
<u>S</u>	Show lines
<u>SS</u>	Show lines in block
<u>SC</u>	Sentence-case lines
<u>SCC</u>	Sentence-case lines in block
<u>SI</u>	Show indentation
<u>T</u>	Select text lines

<u>TI</u>	Select text lines in block
<u>TABS</u>	Display Tabs line
<u>TB</u>	Text Break a line
<u>TBB</u>	Text Break lines in block
<u>TC</u>	Title-case lines
<u>TCC</u>	Title-case lines in block
<u>TF</u>	Text-flow a paragraph
<u>TFF</u>	Text-flow a block of paragraphs
<u>TG</u>	Text glue lines together
<u>TGG</u>	Text glue a block together
<u>TJ</u>	Text join lines together
<u>TJJ</u>	Text join a block together
<u>TL</u>	Trim lines
<u>TLL</u>	Trim lines in block
<u>TM</u>	Set Text Margin in a paragraph
<u>TMM</u>	Set Text Margin in a block of paragraphs
<u>TR</u>	Truncate lines
<u>TRR</u>	Truncate lines in block
<u>TS</u>	Text split a line
<u>TU</u>	Toggle User Line state of lines
<u>TUU</u>	Toggle User Line state of block
<u>TX</u>	Toggle excluded state of lines
<u>TXX</u>	Toggle excluded state of block
<u>UC</u>	Upper-case lines
<u>UCC</u>	Upper-case lines in block
<u>U</u>	Mark User lines
<u>UU</u>	Mark User lines in block
<u>V</u>	Revoke User line status
<u>VV</u>	Revoke User line status in block
<u>W</u>	Swap lines
<u>WW</u>	Swap lines in block
<u>WORD</u>	Display valid WORD characters
<u>X</u>	Exclude lines
<u>XX</u>	Exclude lines in block
<u>{</u>	Column shift left
<u>{{</u>	Column shift left in block
<u>}</u>	Column shift right
<u>}}</u>	Column shift right in block
<u><</u>	Data shift left
<u><<</u>	Data shift left in block
<u>></u>	Data shift right
<u>>></u>	Data shift right in block
<u>[</u>	Indent shift left
<u>[[</u>	Indent shift in block
<u>]</u>	Indent shift right
<u>]]</u>	Indent shift right in block

Rules for Entering Line Commands

Line commands are normally entered into the field on the left of each data line. This field may be referred to as the "line command area" or the "sequence number area".

This field has a default size of six columns, as in ISPF. The size of this field can be adjusted from 5 to 8 columns, to handle special requirements you may have. See [Width of Line numbers \(5-8\)](#) in [Options - Screen](#) for more information.

You may enter a line command by one of the following:

- Type the command in the line command area and press Enter.
- Place the cursor in the data or line command field, and press a keyboard key to which the line command is assigned.
- When a line command is mapped to a key, the command is enclosed in { } braces. See [Keyboard Customization and Keyboard Macros](#) for more information.
- If you try to type the : colon notation for key-mapping of line commands that is utilized in IBM ISPF, it is not directly supported. In prior versions, if you tried it on the command line, you would get an invalid syntax error. For KEYMAP, if you previously mapped **Alt-R** to just :R, when you use it, it would **not** repeat a line but would simply report, **Unknown command**. Recent SPFLite versions will result in a warning popup message advising the user that the command has been automatically converted to the SPFLite-compatible equivalent format of {R}.

When you first start typing a command into the line number area, the cursor can be anywhere within the sequence-number field you see. SPFLite will blank-out the entire area and move the cursor to the left side before entering the first character. That way, there will be no leftover line-number characters to worry about when you key-in a line command, and you don't have to be that precise about where you start typing it.

The following rules apply to all line commands:

- You can type several line commands and make multiple data changes before you press Enter. The editor displays an error message if the line command is ambiguous. Because the line commands are processed from top to bottom, it is possible to have one error message appear that masks a later error condition. Only the first error condition found is displayed. After you have corrected that error condition, processing can continue and the next error condition, if any, is displayed. If you type a line command incorrectly, you can replace it before you press Enter by retyping it, blanking it out, or entering [RESET](#) on the Command Line.
- The Top of Data line will only accept the following line commands:

I [n]	Insert one or n temporary insert lines after this line.
N [n]	Insert one or n permanent lines after this line.
A [n]	Move or copy a line or lines one or n times after this line
AA [n]	Move or copy a line or lines after each nth line of a block
- The Bottom of Data line will only accept the following command:

B [n]	Move or copy a line or lines one or n times before the last data line.
BB [n]	Move or copy a line or lines before each nth line of a block

Note: The notation **[n]** just means the **n** value is optional; you don't actually type the **[** or **]** brackets after the

line command.

Many line commands can take advantage of [Extended Line Command Modifiers](#), as described in the next section.

Note: When a line command is longer than 1 character, the "block mode" name of the command is formed by repeating the last letter. For example, the command **UC** becomes **UCC**.

Extended Line Command Modifiers

Contents of Article

- [*Introduction*](#)
- [*Standard command forms with modifiers*](#)
- [*Retain line-command modifier*](#)
- [*Forward and backward modifiers*](#)
- [*Post-exclude and post-unexclude modifiers*](#)
- [*Coexistence of & and +/- modifiers*](#)
- [*Examples*](#)

Introduction

Many line commands support the use of Extended Line Command Modifiers. These are optional characters appended to the right end of a line command, requesting additional special processing to be performed.

Standard command forms with modifiers

The standard form of a line command with a modifier is to place the modifier after the command **name**, rather than the end of the command as a whole.

For example, if a command to copy 5 lines (**C5**) was to be **retained** with a **&** modifier, the standard form is **C&5** rather than **C5&**. The standard way to lower-case 3 lines and then exclude them following the operation would be with a command **LC-3** rather than **LC3-**.

However, SPFLite also accepts commands with modifiers in non-standard forms. When the display is refreshed, commands are redisplayed in their standard form. If you enter the command **C5&**, it will be redisplayed as **C&5**.

Retain line-command modifier

The addition of the modifier **&** to a line command requests that the command be **retained** on the line following completion of the command rather than having the command disappear when completed. This is similar to the use of **&** prefix on the primary command line to retain the primary command.

This could be used, for example, when copying a range of line commands with a pair of **CC** commands and you will be copying these lines to multiple scattered locations. If the **CC** commands are entered as **CC&** commands and the destination as **A** then following completion of the copy, the **CC&** commands will remain on the selected lines awaiting placement of the next **A** line command for the next copy operation. When all copies are done, the **CC&** can be manually blanked, or cleared with a [**RESET COMMAND**](#) command, usually abbreviated as **RES CMD**.

Note: Once a command is "retained" on a line using **&**, a simple **RESET** command without **CMD** will not clear it.

If you enter a pair of block commands, one with **&** and one without **&**, the command without **&** will be "promoted" to one with the **&** on it. For example, if you enter **CC** on one line and **CC&** on another, the **CC** will be changed into **CC&**.

The **&** line command option can be used with the [**A**](#), [**B**](#), [**C**](#), [**CC**](#), [**H**](#), [**HH**](#), [**Q**](#), [**QQ**](#), [**OR**](#), and [**ORR**](#)

Note: Users of recent versions of IBM ISPF should be aware that SPFLite's implementation of the **&** modifier is different from the ISPF **K** support. The SPFLite retain modifier applies to more commands, and the

semantics are slightly different. SPFLite uses the **&** because SPFLite's support was not really compatible with ISPF, and for us to use **K** was somewhat misleading.

Forward and backward modifiers

For line commands that accept a 'number of lines' operand, the Forward modifier **/** and the Backward modifier **** may be used instead of a number to represent the range of lines starting from the one on which the command is placed to the beginning or end of the dataset. Appending a **/** character requests all lines from the marked line Forward to the last line inclusive. Appending a **** character requests all lines from the marked line Backward to the top of the file inclusive.

For example, entering a **D** on line 3 would request deletion of lines 1 through 3. Entering a **C/** on line 1 would select the entire file as the range to be used for whatever Primary command was entered.

Note: Be careful using the **D** line command with forward or backward modifiers. If you were to place **D/** on line 1 of a file and press Enter, **every line of the file will be deleted**. There are no safeguards, and no "are you sure" messages to prevent you from doing this by accident. If you did issue a **D/** on line 1 and didn't mean to, you can either (a) immediately issue an **UNDO** command, or (b) issue a **CANCEL** command to get out of the edit session without saving the (now zero-length) file. Bear in mind that if you issue a **CANCEL** command, you will lose **all** changes you have made to your file that were done after the last time the file was saved. Because the **UNDO** action may be more helpful than the **CANCEL** action, be sure you have issued an appropriate **SETUNDO** command to enable the **UNDO** command to work as you need it to. The **SETUNDO** status is defined in the Profile and is specific to each file type.

For marking lines using the Forward **/** modifier, from a specific line to the bottom, this is equivalent to using a dummy 99999 operand to mean the rest of the file. IBM ISPF has no notation equivalent to the Backward **** modifier.

The **/** and **** line command options can be used with the [C](#), [D](#), [G](#), [H](#), [J](#), [LC](#), [M](#), [O](#), [OR](#), [R](#), [S](#), [SC](#), [TC](#), [TG](#), [TJ](#), [TR](#), [UC](#), and [X](#)

For **R** and **TR** line commands with **/** or **** modifiers, SPFLite converts these to an "equivalent block form" of **RR** and **TRR**, and has nothing to do with repetition factors or line lengths. See the [R / RR - Repeat Lines](#) and [TR / TRR - Trim Trailing Blanks](#) commands for more information.

The Pad to Length line command **PL** will accept a special modifier of **/** or ****. When used in this way, a command of **PL/** will pad all following lines to a minimum length of 1; **PL** will do the same to preceding lines. Placing **PL/** on line 1 of a file can be used to ensure that all lines in the file have a minimum line length of 1; that is, it is a quick way to ensure there are no zero-length lines in the file. If you need to pad to a longer length (such as 4 for example), you can use the block mode version with **PLL4** on line 1 and **PLL** on the last line.

See [PL / PLL - Pad Lines](#) for more information.

Note: Re: foreign keyboards. Since the **/** and **** characters are shifted characters on many keyboards, SPFLite will accept a period **(.)** in place of a **/** and two periods **(..)** in place of a **** to simplify typing.

Post-exclude and post-unexclude modifiers

The addition of one of **+** or **-** to a line command requests the lines specified by the command be excluded **(-)** or unexcluded **(+)** following the operation. For example, using **CC-** instead of **CC** for a copy operation requests the original lines being copied be excluded following the operation, just as if the range had been marked by **XX** line commands. If you used a regular **CC** block, and copied lines to an A- line, the original lines would not be excluded, but the new copies of those lines would be excluded.

(See also comments below about commands such as **A&-**).

The + and - line command options can be used with the [A](#), [AA](#), [B](#), [BB](#), [C](#), [CC](#), [H](#), [HH](#), [LC](#), [LCLC](#), [LCC](#), [O](#), [OO](#), [OR/ORR](#), [R](#), [RR](#), [SC](#), [SCSC](#), [SCC](#), [TC](#), [TCTC](#), [TCC](#), [TG](#), [TJ](#), [UC](#), [UCUC](#), [UCC](#), [\(](#), [\(](#), [\)](#), [\)\)](#)

Coexistence of & and +/- modifiers

The & (Retain) line command modifier, and the -/+ (post-exclude and post-unexclude) line command modifier, can coexist for the line commands **A**, **B**, **C/CC**, **O/OO** and **OR/ORR**. In **CC**, **OO** and **ORR** the lines that begin and end the block will not themselves be excluded or unexcluded by the -/+ modifier, but only the lines *within* those blocks.

Why would you wish to have both a & and a + or - modifier at the same time? Consider an example where you might to gather a copy of several different blocks of lines, scattered throughout your edit file, and you want them copied to some fixed point, and you also want those lines to be initially excluded so they don't distract you. You could set up an **A&-** or **B&-** line command as the "sink" for those **CC** or **MM** line commands.

Note that if you wanted all of the moved or copied lines to get copied to the kept copy point, and if you used an **A&-** for this, each group of lines would be copied just *after* the **A&-** line, which means they would end up being just *before* the last group of lines copied in. So, if you used **A&-** for that purpose, each successive group of lines would get copied in *reverse order*, much like a 'stack'. For most users, that will be a surprising and undesired effect. To get the lines copied in the same order as you issue the **CC** or **MM** line commands, the copy-point should be defined as **B&-** instead. A little trial and experimentation will make this point clear. Once your copying or moving of lines is complete, just go back and remove the persistent **A&-** or **B&-** command from the line it was on.

Examples

Copy 3 lines after another and exclude the copied lines following the operation.

```
C3      AAA.....
000002 BBB.....
A-
000004 DDD.....
```

would result in

```
000001 AAA.....
000002 BBB.....
000003 CCC.....
----- < 000003 > -----
000007 DDD.....
```

Copy a line and after keep the copy command for re-use.

```
C&      AAA.....
000002 BBB.....
A
000004 DDD.....
```

would result in

```
C&      AAA.....
000002 BBB.....
000003 CCC.....
000004 AAA...
000005 DDD.....
```

A - After Destination

Syntax

A [n]

Operands

n A number that tells the editor to repeat the associated line command a specified number of times. If you do not type a number, or if the number you type is 1, the editor performs the command only once.

The number does not affect associated primary commands.

Description

To specify that data is to be moved, copied, or inserted after a specific line. The After command **A** is used in conjunction with one of the line commands [C / CC - Copy Lines](#) or [M / MM - Move Lines](#), or one of the Primary Commands [COPY](#) or [PASTE](#) to specify a destination for the command action.

1. Type **A** in the line command area of the line **after** which the moved, copied, or inserted data is to be placed.
2. If you are specifying the destination for a line command, a number after the **A** line command specifies the number of times the other line command is performed. However, a number after the **A** command has no affect on a primary command (like [COPY](#) or [PASTE](#)); that limitation is required for ISPF compatibility.
3. Press Enter. Some of the primary commands may cause a dialog box to be displayed if more information is needed (like file names). If so, fill in the required information and press Enter to move, copy, or insert the data. Refer to information about the specified command if you need help.
4. If no panel is displayed, the data is moved, copied, or inserted when you press Enter in step 3.

You must always specify a destination when moving or copying data, **except** when you are using a primary command to move, copy, or insert data into a file that is **empty**. In such cases, the empty file is treated as if an **A** line command were present on the Top of Data line.

AA - After Block

Syntax

```
AA[n] / AA
```

Operands

n A number that tells the editor to repeat the associated line command or [COPY](#) or [PASTE](#) command after every *nth* line in the AA block. If you do not type a number, or if the number you type is 1, the editor performs the command after each line in the range.

Description

The **AA** block command is used to copy or insert after every line in a block of lines. If a line or group of lines is to be inserted after (or before) every line in a range of lines, standard ISPF requires the inserted data to be manually placed by individual copy and **A** (or **B**) line commands. SPFLite will allow definition of an **AA** block, and then insertion of data will occur after every line in that block.

If a number **n** follows the **AA**, it means to copy after every **n**th line. If **n** is omitted it defaults to a value of 1, meaning 'after every line'.

The value of **n** is used to determine the insertion points where moved or copied lines are to be placed. Insertion points are determined by going forwards and treating the initial **AA** block command as the starting point.

The examples below are shown using a **M** line command. **AA** can be used with **C/CC** or **M/MM** or the Primary commands [COPY](#) and [PASTE](#) as well.

As an example, consider an **AA** block. Assume that the first **AA** command is on "relative line 1". If **n** is omitted, then lines are copied or moved after every line in the block:

Before:

```
***** **** Top of Data ****
M          PIC X(1).
AA
AA-
***** **** Bottom of Data ****
```

After:

AA_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\AA_Block.TXT

File Manager AA_Block.TXT

Command >

```
***** **** Top of Data ****
000001 - 05 ONE
000002      PIC X(1).
000003 - 05 TWO
000004      PIC X(1).
000005 - 05 THREE
000006      PIC X(1).
***** **** Bottom of Data ****
```

Edit * L 000001 C 1 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0011

If **n** is present, then lines are copied or moved after “relative line **n**”, “relative line **2n**”, “relative line **3n**”, etc. Here, note that the first **AA** command is on line 2, which is “relative line 1”, and “relative line **n**” where **n**=2 is line 3. So, the insertion points are after lines 3, 5 and 7:

Before:

AA_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\AA_Block.TXT

File Manager AA_Block.TXT

Command >

```
***** **** Top of Data ****
M -----
AA2      05 ONE
000003      PIC X(1).
000004 - 05 TWO
000005      PIC X(1).
000006 - 05 THREE
AA_      PIC X(1).
***** **** Bottom of Data ****
```

Edit * L 000007 Lines: 7 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0020

After:

AA_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\AA_Block.TXT

File Manager AA_Block.TXT

Command >

```
***** **** Top of Data ****
000001 - 05 ONE
000002      PIC X(1).
000003 *-----
000004 - 05 TWO
000005      PIC X(1).
000006 *-----
000007 - 05 THREE
000008      PIC X(1).
000009 *-----
***** **** Bottom of Data ****
```

Edit * L 000001 C 1 Lines: 9 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0011

B - Before Destination

Syntax

B [n]

Operands

- n** A number that tells the editor to repeat the associated line command a specified number of times. If you do not type a number, or if the number you type is 1, the editor performs the command only once.

The number does not affect associated primary commands.

Description

To specify that data is to be moved, copied, or inserted **before** a specific line. The **B** - Before command is used in conjunction with one of the line commands [C / CC - Copy Lines](#) or [M / MM - Move Lines](#), or one of the Primary Commands [COPY](#) or [PASTE](#) to specify a destination for the command action.

1. Type **B** in the line command area of the line **before** which the moved, copied, or inserted data is to be placed.
2. If you are specifying the destination for a line command, a number after the **B** line command specifies the number of times the other line command is performed. However, a number after the **B** command has no affect on a primary command (like [COPY](#) or [PASTE](#)); that limitation is required for ISPF compatibility.
3. Press Enter. Some of the primary commands may cause a dialog box to be displayed if more information is needed (like file names). If so, fill in the required information and press Enter to move, copy, or insert the data. Refer to information about the specified command if you need help.
4. If no panel is displayed, the data is moved, copied, or inserted when you press Enter in step 3.

You must always specify a destination when moving or copying data, **except** when you are using a primary command to move, copy, or insert data into a file that is **empty**.

BB - Before Block

Syntax

```
BB[n] / BB
```

Operands

n A number that tells the editor to repeat the associated line command or [COPY](#) or [PASTE](#) primary command before every *nth* line in the BB block. If you do not type a number, or if the number you type is 1, the editor performs the command before each line in the range.

Description

The **BB** block command is used to copy or insert before every line in a block of lines. If a line or group of lines is to be inserted after (or before) every line in a range of lines, standard ISPF requires the inserted data to be manually placed by individual copy and **A** (or **B**) line commands. SPFLite will allow definition of a **BB** block, and then insertion of data will occur before every line in that block.

If a number **n** follows the **BB**, it means to copy before every **n**th line. If **n** is omitted it defaults to a value of 1, meaning before every line'.

The value of **n** is used to determine the insertion points where moved or copied lines are to be placed. Insertion points are determined by going backwards and treating the final **BB** block command as the starting point.

The examples below are shown using a **M** line command. **BB** can be used with **C/CC** or **M/MM** or the Primary commands [COPY](#) and [PASTE](#) as well.

As an example, consider a **BB** block. Assume that the last **BB** command is on "relative line 1". If **n** is omitted, then lines are copied or moved before every line in the block, working backward:

Before:

```
BB_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\BB_Block.TXT
File Manager BB_Block.TXT
Command > Scroll > HALF
*****
BB          PIC X(1).
000002      PIC X(1).
BB          PIC X(1).
M 05 FILLER
*****
Edit * L 000004 Lines: 4 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0014
```

After:

BB_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\BB_Block.TXT - - X

File Manager BB_Block.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001    05 FILLER
000002          PIC X(1).
000003    05 FILLER
000004          PIC X(1).
000005    05 FILLER
000006          PIC X(1).
***** **** Bottom of Data ****
```

Edit * L 000002 C 1 Lines: 6 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0025

If **n** is present, then lines are copied or moved before “relative line **n**”, “relative line **2n**”, “relative line **3n**”, etc. working backward. Here, note that the last BB command is on line 8, which is “relative line 1”, and “relative line **n**” where **n**=2 is line 6. So, the insertion points are before lines 4, 6 and 8.

Before:

BB_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\BB_Block.TXT - - X

File Manager BB_Block.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
BB2| 05 FILLER
000002          PIC X(1).
000003 05 FILLER
000004          PIC X(1).
000005 05 FILLER
BB             PIC X(1).
M
***** **** Bottom of Data ****
```

Edit * L 000001 Lines: 7 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0014

After:



BB_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\BB_Block.TXT

File Manager BB_Block.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 *-----+
000002 | 05 FILLER
000003 |          PIC X(1).
000004 *-----+
000005 | 05 FILLER
000006 |          PIC X(1).
000007 *-----+
000008 | 05 FILLER
000009 |          PIC X(1).
***** **** Bottom of Data ****
```

Edit * L 000002 C 1 Lines: 9 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0014

BNDS - Display Bounds Line

Syntax

```
BNDS | BND
```

Operands

None

Description

The **BNDS** line command provides an alternative to setting the boundaries with the [BOUNDS](#) primary command; the effect on the data is the same. However, if you use both the **BOUNDS** primary command and the **BNDS** line command in the same interaction, the Primary command overrides the Line command.

The **BNDS** definitions remain in effect, even if they are not displayed, until you change them. **BNDS** definitions are retained as part of the file type PROFILE, and are automatically used the next time you edit the same kind of data (based on the file extension).

To display the boundary definition (=BNDS>) line:

1. Type **BNDS** in the line command area of any unflagged line.
2. Press Enter
3. The boundary definition line is inserted.

To change the BOUNDS settings:

Note: You can use the [COLS](#) line command with the **BNDS** line command to help check and reposition the BOUNDS settings. The **COLS** line command displays the column identification line.

1. Delete a < or > or + marker. The < marker shows the left BOUNDS setting and the > marker shows the right BOUNDS setting. The + marker indicate the Right bound is to be equal to the largest current record length, this is the default.
2. Move the cursor to a different location on the =BNDS> line.
3. Retype the deleted marker or markers. Note: The < character must be typed to the left of the > or + character.
4. Press Enter.
5. The new BOUNDS settings are now in effect.

To revert to the default settings:

1. Display the boundary definition line. Blank out its contents with the Erase EOF key, the cursor, or the Del (delete) key.
2. Press Enter.
3. The default boundary settings will take effect.

To remove the boundary definition line from the display:

1. You can either type [D](#) in the line command area that contains the =BNDS> flag or type [RESET](#) on the Command line.
2. Press Enter.
3. The =BNDS> line is removed from the display.

Note: If you delete the =BNDS> line, it merely gets rid of the **display** of the bounds information on that line. The bounds **specification** defined by that =BNDS> line is **still** in effect. If you want to get rid of the

specification, apply the steps to "revert the default settings.

A word about the use of BOUNDS

The BOUNDS feature of ISPF is one that IBM did not extensively document, and in practice, mainframe ISPF users do not tend to use this feature very often. Every effort was made to implement BOUNDS in SPFLite in an ISPF-compliant manner. However, it may produce surprising and unexpected results if you are not familiar with the actions taken by various commands when operating under restricted column BOUNDS. The "surprising and unexpected" aspect is even more of a factor for SPFLite users without a prior mainframe ISPF background.

For many users, you will likely get the most benefit from SPFLite by operating in unbounded mode most or all of the time, and not worry about using BOUNDS unless you have very particular editing requirements.

Note: When the **BOUNDS** setting is anything other than **MAX**, the status line display will show the **BOUNDS** setting in white letters on a red background, like **Bnds: 1 to 40** so that it can't be ignored. This will help users to avoid the unexpected and nonstandard handling of data that occurs when non-default bounds are in effect, if that was not their intent.

C / CC - Copy Lines

Syntax

C[n]

CC / CC[n]

Operands

n

The number of lines to be copied. If you do not type a number, or if the number you type is 1, only the line on which you type **C** is copied

Description

Note: The **C/CC** and **M/MM** line commands are also used to mark lines for the use of various primary commands. When used for this purpose, you may **not** also enter other non primary-related line commands during the same interaction. e.g. if using **CC** block to mark a range of lines for use by the **CREATE** primary command, you may not, **at the same time**, enter other line commands such as **I** Insert, **D** Delete, etc.

When used to "mark off" a line range for use by a primary command, **CC** is usually the line command of choice. **MM** can also be used, but it causes the marked range of lines to be deleted after the primary command is performed.

Note: In order to be ISPF compatible, the **FIND** command will not use a pending **C/CC** or **M/MM** block to define a line range, because of the long-standing custom of ISPF users to keep such blocks pending while using **FIND** to search for a place into which to move or copy that block. To use **FIND** and have a **C/CC** or **M/MM** block to define a line range, use the **FIND** alias of **FF**.

See [FIND - Find a Character String](#) for more information.

To copy one or more lines within the same file:

1. Type **C** in the line command area of the line to be copied. If you also want to copy one or more lines that immediately follow this line, type a number greater than 1 after the **C** command.
2. Next, specify the destination of the line to be copied by using either the [A](#) (after), [AA](#) (Block After), [B](#) (before), [BB](#) (Block Before), [O](#) (overlay) or [H/HH](#) (Here) line command.
3. Press Enter.
4. The line or lines are copied to the new location.

To copy a block of lines within the same file:

1. Type **CC** in the line command area of both the first and last lines to be copied. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **CC** and the second **CC**, if necessary.
2. Use the [A](#) (after), [AA](#) (Block After), [B](#) (before), [BB](#) (Block Before), [OO](#) (overlay) or [H/HH](#) (Here) command to show where the copied lines are to be placed. Notice that when you use the block form of the **C** command (**CC**) to copy and overlay lines, you should also use the block form of the **O** command (**OO**).
3. Press Enter
4. The lines that contain the two **CC** commands and all of the lines between them are copied to the new location.

COLS - Display Columns Line

Syntax

```
COLS
```

Operands

None

Description

The **COLS** line command will display the column-position (=COLS>) line, which looks like:

```
=COLS> -----1-----2-----3-----4-----5
```

When there are TABS in effect, each tab column will be shown in the COLS display as an underscore.

Suppose you had Tabs set every 5 columns. When you displayed the COLS line, it would look like this:

```
=COLS> -----_-----1-----_-----2-----_-----3-----_-----4-----_-----5
```

To display the column-position line:

1. Type **COLS** in the line command area of any line.
2. Press Enter.
3. The column-position line is inserted.

Note: You can use the **COLS** line command with the **BNDS** line command to help check and reposition the bounds settings.

To remove the column-position line from the panel:

1. You can either type **D** in the line command area that contains the =COLS> flag or type **RESET** on the Command line.
2. Press Enter.
3. The =COLS> line is removed from the display.

SPFLite also supports a **Primary** command **COLS** which displays a **COLS** line at the top of the screen at all times. To display this line, use the **COLS** Primary command.

D / DD - Delete Lines

Syntax

```
D [n]  
DD / DD
```

Operands

n The number of lines to be deleted. If you do not type a number, or if the number you type is 1, only the line on which you type **D** is deleted

Description

To delete one or more lines:

1. Type **D** in the line command area of the line to be deleted. If you also want to delete one or more lines that immediately follow this line, type a number greater than 1 after the **D** command.
2. Press Enter.
3. The line or lines are deleted.

To delete a block of lines:

1. Type **DD** in the line command area of both the first and last lines to be deleted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **DD** and the second **DD**, if necessary
2. Press Enter.
3. The lines that contain the two **DD** commands and all of the lines between them are deleted.

Use of D line command in Multi-Edit sessions

If you use the **D** line command to delete a **=FILE>** marker in a Multi-Edit session, it **will detach that file** from the session. Doing so removes the **=FILE>** marker and all data lines after it that are associated with *that file*, but it **will not delete the underlying file** on disk.

F - Display First Lines in Excluded Range

Syntax

```
F[n]
```

Operands

n The number of lines to be redisplayed. If you do not type a number, or if the number you type is 1, only one line is redisplayed.

Note that the / forward modifier and the \ backward modifier are not allowed on the **F** command in place of **n**.

To redisplay an entire excluded region, use the [S](#) line command.

Description

To redisplay the first **line** or lines of a block of excluded lines:

1. Type **F** in the line command area next to the dashed line that shows where lines have been excluded (the excluded-line placeholder). The message in the dashed line tells you how many lines are excluded. If you want to redisplay more than one line, type a number greater than 1 after the **F** command.
2. Press Enter
3. The first line or lines are redisplayed.

Note: Also see the [L](#), [S/SS](#) and [SI](#) line commands for alternative re-display commands.

G / GG - Glue Lines Together

Syntax

```
G[n]  
GG / GG
```

Operands

- n** The number of lines to be glued. If you do not type a number, or if the number you type is 1, the line on which the **G** is entered will be joined to the line following it.

Description

G/GG is used to glue together one or more lines as 'physical' lines, without being trimmed beforehand. Any leading or trailing spaces on the lines are retained. The lines are joined into a single line by concatenating them together left to right, in order.

Note: To perform a Glue operation that involves trimming, see the line command [TG / TGG - Text Glue Lines](#).

By default, nothing is inserted between the glued lines; they are simply concatenated together. In effect, lines are glued together with an implied zero-length string between them.

In some cases, it may be useful to specify a particular user-defined string to insert between lines. This is now possible using the **GLUEWITH** primary command. See the **GLUEWITH** command, and the example below, for more information. Note that **GLUEWITH** is a global setting, not associated with a particular file type.

Gluing of lines uses the following rules:

- The lines are treated as raw data
- Blanks are neither added nor removed during the Glue process
- No space or other data is added following the last character of each line before appending the next line, unless **GLUEWITH** is in effect.

Example 1: Before Glue:

Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

File Manager Glue.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
000004 LINE FOUR.
GG LINE FIVE.
000006 LINE SIX.
GG LINE SEVEN.
000008 LINE EIGHT.
***** **** Bottom of Data ****
```

Edit | L 000007 | Lines: 8 | Cols 1 to 80 | Bnds: MAX | INS | T | CS | S+ | Line Len 0012 |

After Glue:

Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

File Manager Glue.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
000004 LINE FOUR.
000005 LINE FIVE. LINE SIX. LINE SEVEN.
000006 LINE EIGHT.
***** **** Bottom of Data ****
```

Edit * | L 000005 | Lines: 6 | Cols 1 to 80 | Bnds: MAX | INS | T | CS | S+ | Line Len 0037 |

Example 1: Before Glue, **GLUEWITH '---'** in effect:

Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

File Manager Glue.TXT | Command > Scroll > HALF
GLUEWITH set to '---'

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
000004 LINE FOUR.
GG LINE FIVE.
000006 LINE SIX.
GG LINE SEVEN.
000008 LINE EIGHT.
***** **** Bottom of Data ****
```

Edit | L 000007 | Lines: 8 | Cols 1 to 80 | Bnds: MAX | INS | T | CS | S+ | Line Len 0012 |

After Glue:

Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

File Manager Glue.TXT

Command >

***** **** Top of Data *****

000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
000004 LINE FOUR.
000005 LINE FIVE. --- LINE SIX. ---LINE SEVEN.
000006 LINE EIGHT.
***** **** Bottom of Data *****

Edit * L 000005 Lines: 6 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0046

H / HH - HERE Destination

Syntax

```
H[n]
HH[n] / HH
```

Operands

n A number that tells the editor to repeat the associated line command a specified number of times. If you do not type a number, or if the number you type is 1, the editor applies the command to a single line or block.

Description

The **H** command, which specifies 1 line, or the **HH / HH** pair, which mark a range of lines, specify lines which are to be **replaced** by the associated data selected by line commands ([C](#) - Copy or [M](#) - Move). This is similar to using the [A](#) or [B](#) line commands to mark the destination for a move/copy except in addition the lines marked with **H / HH** are **replaced** by the copied/moved data.

Note: ISPF users who do not have the **H/HH** command may be in the habit of simulating an **H/HH** by issuing a **C/CC** or **M/MM**, and then when they navigate to the place where they want the new lines to replace the old ones, they may have used a combination of **A** and **D**, or perhaps they used the Erase EOF key to clear the destination lines and then use an **O/OO** overlay command. In comparison, the SPFLite method using **H/HH** for this purpose is much simpler.

To help remember this command

A = **A**fter here
 B = **B**efore here
 H = *right* **H**ere

Using the H/HH command

1. Mark the lines to be replaced with an **H/HH** line command.
2. Any number specified on the **H/HH** line command itself indicates the number of times the other line command is performed.
3. Press Enter.
4. The data is moved or copied, replacing the line(s) marked with the **H/HH** line commands.

Other line commands that are used to specify a destination are the [A](#) (After) command, [B](#) (Before) command, [O](#) (Overlay) command, [OR](#) (Overlay-Replace) command, and the [AA](#) and [BB](#) block-destination commands.

I - Insert New (Temporary) Blank Lines

Syntax

I [n]

Operands

n The number of blank lines to insert. If you do not type a number, or if the number you type is 1, only one line is inserted.

Description

The **I** line command inserts one or more lines in your file. The inserted lines are blank, unless you have defined a non-blank mask. See [MASK - Set the Insert line mode](#) for more information about defining a mask. By default, the MASK line is defined as blank, and when that is the case, lines inserted with the **I** line command are blank.

Technically, for any given file type, there is **always** a **MASK** definition present. The only question is whether it is blank or not.

1. Type **I** in the line command area of the line that the inserted line is to follow. If you want to insert more than one line, type a number greater than 1 after the **I** command.

It is legal to ask for more temporary lines than the screen can display, but these will disappear when you attempt to scroll the screen.

2. Press Enter.
3. The line or lines are inserted.

If you type any information (even the spacebar) on a temporary line, the line becomes a permanent part of the data and is assigned a line number the next time you press Enter. However, if you do not type any information, the space for the new line is automatically deleted the next time you press Enter.

If you do not type any information on a temporary line when a non-blank MASK is defined, the non-blank characters of the MASK are not considered to be "entered data". If you press Enter without typing any characters of your own, the temporary line containing these non-blank MASK characters is not retained.

If you type information on the last, or only, inserted line and the cursor is still in the data portion of that line, the editor automatically inserts another line when you press Enter. That way you can continue entering new data without having to re-issue the **I** command each time.

Note: IBM ISPF inserts MASK lines when you first edit a new file. SPFLite does not implement that action, nor does it initially fill the screen with temporary blank lines of any kind for a new file. To get that effect, you must manually enter an **I** line command yourself. If you wanted to achieve the effect of completely filling an edit screen with temporary blank lines, you can issue a line command like **I99** on the Top of Data line, or at the top-most line being displayed. SPFLite will fill-out the displayed screen with blank lines, as many as will fit, and any "extras" that don't fit are simply ignored.

See also the **N** command for inserting permanent blank lines. The **N** line command does **not** utilize the MASK line.

J / JJ - Join Lines Together

Syntax

J[n]
JJ / JJ

Operands

n The number of lines to be joined. If you do not type a number, or if the number you type is 1, the line on which the **J** is entered will be joined to the line following it.

Description

J/JJ is used to join together one or more lines as 'physical' lines, without being trimmed beforehand. Any leading or trailing spaces on the lines are retained. The lines are joined into a single line by concatenating them together left to right, in order, with a single blank character in between them.

Note: To perform a Join operation that involves trimming, see the line command **TJ / TJJ - Text Join Lines**.

A single blank character is inserted between the joined lines. To insert something else between the lines, it is necessary to use the **Glue** line commands in conjunction with the **GLUEWITH** primary command. Joining of lines uses the following rules:

- The lines are treated as raw data.
 - Blanks are neither added nor removed during the Join process, except that a single space character is added following the last character of each line before appending the next line.
 - The **GLUEWITH** string is not considered, even if defined.

Example 1: Before Join,

After Join:



File Manager Join.TXT

Command >

Scroll > HALF

```
***** ***** Top of Data *****  
000001 LINE ONE.  
000002 LINE TWO.  
000003 LINE THREE.  
=COLS> -----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8  
000004 LINE FOUR.  
000005 LINE FIVE. LINE SIX. LINE SEVEN.  
000006 LINE EIGHT.  
***** ***** Bottom of Data *****
```

Edit * L 000005

Lines: 6

Cols 1 to 80

Bnds: MAX

INS

T

CS

S+

Line Len 0039

L - Display Last Lines in Excluded Range

Syntax

```
L [n]
```

Operands

n The number of lines to be redisplayed. If you do not type a number, or if the number you type is 1, only one line is redisplayed.

Note that the / forward modifier and the \ backward modifier are not allowed on the **L** command in place of **n**. To redisplay an entire excluded region, use the [S](#) line command.

Description

To redisplay the **last** line or lines of a block of excluded lines:

1. Type **L** in the line command area next to the dashed line that shows where lines have been excluded (the excluded-line placeholder). The message in the dashed line tells you how many lines are excluded. If you want to redisplay more than one line, type a number greater than 1 after the **L** command.
2. Press Enter
3. The last line or lines are redisplayed.

Note: See also the [F](#), [S/SS](#) and [SI](#) line commands for alternative re-display commands.

LC / LCC - Lower-Case Lines

Syntax

LC [n]	LCC is the block form of this command.
LCC / LCC	

Operands

n	The number of lines to be lower-cased. If you do not type a number, or if the number you type is 1, only the line on which you type LC is lower-cased
----------	--

Description

To convert characters on one or more lines to lower-case:

1. Type **LC** in the line command area of the line that contains the characters you want to convert. If you also want to convert characters on one or more lines that immediately follow this line, type a number greater than 1 after the **LC** command.
2. Press Enter.
3. The characters on the specified line(s) are converted to lower-case.

To convert characters in a block of lines to lower-case:

1. Type **LCC** (or **LCLC**) in the line command area of both the first and last lines that contain characters that are to be converted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **LCC** and the second **LCC**, if necessary.
2. Press Enter.
3. The characters in the specified lines that contain the two **LCC** commands and in all of the lines between them are converted to lower-case.

M / MM - Move Lines

Syntax

```
M[n]  
MM / MM[n]
```

Operands

n

The number of lines to be moved. If you do not type a number, or if the number you type is 1, only the line on which you type **M** is moved. For the **MM** command, *n* is the number of copies of the **MM** block to be moved, which is assumed to be 1 if omitted.

The *n* operand cannot be used on the **MM** command if it is participating in a line swap with a **WW** block.

Description

Note: The **C/CC** and **M/MM** line commands are also used to mark lines for the use of various primary commands. When used for this purpose, you may **not** also enter other non primary related line commands during the same interaction. e.g. if using **CC/CC** to mark a range of lines for use by the **CREATE** primary command, you may not, **at the same time**, enter other line commands such as **I** Insert, **D** Delete, etc.

When used to "mark off" a line range for use by a primary command, **CC** is usually the line-command of choice. **MM** can also be used, but it causes the marked range of lines to be deleted after the primary command is performed.

Note: In order to be ISPF compatible, the **FIND** command will not use a pending **C/CC** or **M/MM** block to define a line range, because of the long-standing custom of ISPF users to keep such blocks pending while using FIND to search for a place into which to move or copy that block. To use FIND and have a **C/CC** or **M/MM** block to define a line range, use the FIND alias of **FF**.

See [FIND - Find a Character String](#) for more information.

M/MM can also be used in conjunction with **WW** to swap two blocks of lines. When a swap is performed, the two blocks need not contain the same number of lines.

To move one or more lines within the same file:

1. Type **M** in the line command area of the line to be copied. If you also want to move one or more lines that immediately follow this line, type a number greater than 1 after the **M** command.
2. Next, specify the destination of the line(s) to be moved by using either the **A** (after), **AA** (After Block), **B** (before), **BB** (Before Block) or **O** (overlay) line command. To swap the destination lines with the lines marked by **M**, use the **W** or **WW** line commands on the destination.
3. Press Enter.
4. The line or lines are moved to the new location. If the destination was marked by **W** or **WW**, the lines formerly at the destination are now moved to where the **M** lines had been.

To move a block of lines within the same file:

1. Type **MM** in the line command area of both the first and last lines to be copied. You can scroll (or use **FIND** or **LOCATE**) between typing the first **MM** and the second **MM**, if necessary.
2. Use the **A** (after), **AA** (After Block), **B** (before), **BB** (Before Block) or **OO** (overlay) command to show where the moved lines are to be placed. Notice that when you use the block form of the **M** command (**MM**) to move and overlay lines, you should also use the block form of the **O** command (**OO**). To swap the destination lines with the lines marked by **MM**, use the **W** or **WW** line commands on the destination.
3. Press Enter.
4. The lines that contain the two **MM** commands and all of the lines between them are copied to the new location. If the destination was marked by **W** or **WW**, the lines formerly at the destination are now moved to where the **MM** lines had been.

Note: See [Word Processing Support](#) for information about text swaps.

Example of using **M/MM** and **W/WW** to swap lines:

Before:

MoveSwap.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MoveSwap.TXT

File Manager MoveSwap.TXT

Command > Scroll > HALF

```
***** * Top of Data *****  
MM LINE ONE.  
MM LINE TWO.  
000003 STUFF  
000004 THINGS  
WW LINE FIVE.  
WW LINE SIX.  
***** * Bottom of Data *****
```

Edit * L 000006 Lines: 6 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0010

After:

MoveSwap.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MoveSwap.TXT

File Manager MoveSwap.TXT

Command > Scroll > HALF

```
***** * Top of Data *****  
000001 LINE FIVE.  
000002 LINE SIX.  
000003 STUFF  
000004 THINGS  
000005 LINE ONE.  
000006 LINE TWO.  
***** * Bottom of Data *****
```

Edit * L 000001 Lines: 6 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0010

MARK - Set Column Markers

Syntax

```
MARK
```

Operands

None

Description

When you type **MARK** in the line command area, =MARK> is displayed along with any previously defined column marker positions. To remove the =MARK> line, use the [D](#) (delete) line command or the [RESET](#) primary command, or end the edit session. The =MARK> line is never saved as part of the data.

Column markers are faint vertical lines visible on the screen at specified columns. The column marker definitions remain in effect, even if the =MARK> line is not displayed, until you change them by re-displaying the =MARK> line and adding or removing < > or * characters on it.

When a < code is used on the MARK line, the faint vertical line appears to the left of the < sign, like |<. When a > code is used on the MARK line, the faint vertical line appears to the right of the > sign, like >|.

It may be seen that the < and > codes “point” to the side of the column where the column marker line is to appear, and thus are easy to remember. It is legal to have > and < next to each other as in >< on the same MARK line, like >|<.

The * code from prior versions of SPFLite remains available, and works the same as the < code.

MARK definitions are retained in the PROFILE, and are automatically used the next time you edit files of the same type.

See [Defining Tabs and Column Markers](#) for more information on using Column Markers.

See [Options - Screen](#) for setting the color used for the column markers.

Here is an example of how **MARK** will display these column marker lines:

File Manager **Mark.TXT**

Command >

Scroll > HALF

```
***** ***** ***** ***** ***** ***** Top of Data *****  
000001 LINE FIVE.          Data 5 |  
000002 LINE SIX.          Data 6 |  
000003 STUFF               |  
000004 THINGS              |  
000005 LINE ONE.           Data 1 |  
000006 LINE TWO.           Data 2 |  
=MARK> *                  < >  < >  
***** ***** ***** ***** ***** ***** Bottom of Data *****
```

Edit * | L --- C 2

Lines: 6

Cols 1 to 80

Bnds: MAX

INS

T

CS

S+

MASK - Set the Insert line model

Syntax

```
MASK
```

Operands

None

Description

The **MASK** line command displays the **=MASK>** line. On this line, you can type characters that you want to have automatically inserted into a file. These characters, which are called the *mask*, are inserted whenever you use the **I** (insert) or **TS** (text split) line commands.

Notes:

- The SPFLite **N** line command *always* inserts permanent blank lines that are completely blank, and it will **not** use the definition of the **MASK** line when it creates those blank lines.
- IBM ISPF also inserts **MASK** lines when you first edit a new file. SPFLite does not implement that action, nor does it initially fill the screen with temporary blank lines of any kind for a new file. To get that effect, you must manually enter an **I** line command yourself. If you wanted to achieve the effect of completely filling an edit screen with temporary blank lines, you can issue a line command like **I99** on the Top of Data line, or at the top-most line being displayed. SPFLite will fill-out the displayed screen with blank lines - as many as will fit - and any "extras" that don't fit are simply ignored.

The default initial value for the mask line is blank. Whenever you create a new PROFILE for a new file type, the mask value is always initially set to blank.

To define a non-blank mask:

- Enter or update any characters you wish on the the **=MASK>** line when it is being displayed.
- Press Enter. The mask is now defined.

Once a mask is defined, the contents of the **=MASK>** line are displayed whenever a new line is inserted. This occurs when you use the **I** (insert) and **TS** (text split) line commands. You can change the mask definition whenever you wish by repeating the preceding steps.

The mask line is not saved as part of the data nor as part of the **STATE** information, since it is associated with a file's type as part of the PROFILE, and is not stored with any individual data file.

The mask remains in effect whether it is being displayed or not.

When you type **MASK** in the line command area, **=MASK>** is displayed in the sequence area, along with any previously defined mask line.

To remove the **=MASK>** line, use the **D** (delete) line command or the **RESET** primary command, or end the edit session. When you delete the MASK with a D line command or with a **RESET** primary command, you are merely causing the *display* of the **MASK** line to go away; it does not remove the definition of the **MASK** line itself.

Technically, for any given file type, there is **always** a **MASK** definition present. The only question is whether it is blank or not.

The **=MASK>** line data is saved as part of the PROFILE data for the file type.

The MASK line provides a model text line which will be used to fill in newly inserted lines in the edit session created by the `I` or `TS` line commands.

Using inserted lines with MASK data

The data you define on the MASK line acts like a "prototype" or "overlay" of what each new inserted line will look like. A common use for MASK lines is to insert comments on one side of a source program as new lines of code are written.

It is necessary that you type some original data on such inserted lines, in addition to what the MASK line specifies, even when the MASK line is non-blank. If you don't, SPFLite will assume it's a temporary "blank" line that you don't want, and it will remove it. Any non-blank characters on a MASK line are **not** considered original data in and of itself.

Example

If you request display of the MASK line, and fill it in with the follows:

```
/*
```

then when new lines are inserted, those inserted lines would be initialized using the provided MASK prototype line, like this:

File Manager MaskSample.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001 Data Line 1
000002 Data Line 2
000003 Data Line 3
    /* */ 
    /* */ 
    /* */ 
000007 Data Line 4
000008 Data Line 5
000009 Data Line 6
***** **** Bottom of Data ****
```

Edit * L 000004 C 1 Lines: 9 Cols 1 to 80 Bnds: MAX INS T CS S+ Line Len 0037

MD / MDD - Make Data Line

Syntax

```
MD [n]
MDD / MDD
```

Operands

n The number of lines to be converted to data lines. If n is omitted or specified as zero, one line is converted. The n operand may also be specified as the / forward or \ backward modifier.

Description

The **MD** line command converts a **=NOTE>** line into a normal data line. In addition, special lines that are displayed by the PROFILE command can also be converted into normal data lines. Such lines include **=PROF>**, **=WORD>**, **=MARK>**, **=TABS>**, **=COLS>** and **=BNDS>**.

On the Top of Data and Bottom of Data lines, **MD** is not allowed. On excluded lines, the lines remain excluded after they are converted. It is legal to use **MD** or **MDD** on a line or block in which some or all of the lines are already normal data lines. If this is done, the already-normal lines are simply ignored, and remain normal lines.

To convert one or more special or **NOTE** lines to normal data lines:

1. Type **MD** in the line command area on the line that is to be converted. If you want to convert more than one line, type an 'n' value greater than 1 after the MD.
2. Press Enter. The special lines and **NOTE** lines are converted to normal data lines.

To convert a block of special or **NOTE** lines to normal data lines:

1. Type **MDD** in the line command area of both the first and last lines to be converted. You can scroll (or use the **FIND** or **LOCATE** command) between the first **MDD** and the second **MDD**, if necessary.
2. The **LOCATE [NOT] NOTE** command may be used to find the **FIRST**, **LAST**, **PREV** or **NEXT** line which is (or is not) a **=NOTE>** line.
3. The **LOCATE [NOT] SPECIAL** command may be used to find the **FIRST**, **LAST**, **PREV** or **NEXT** line which is [not] a special line, such as is displayed by the PROFILE command.
4. Press Enter. The lines that contain the two **MDD** commands and all lines between them are converted to normal data lines.

Example:

Here, a number of lines have the Make Data command **MD** placed on them:

Result: Notice that the lines that had **MD** line commands placed on them now have line numbers, because they are now regular data lines.

MN / MNN - Make Note Line

Syntax

```
MN [n]
MNN / MNN
```

Operands

n The number of lines to be converted to note lines. If n is omitted or specified as zero, one line is converted. The n operand may also be specified as the / forward or \ backward modifier.

Description

The **MN** line command converts a normal data or special lines into **=NOTE>** lines. This is the opposite of the action performed by the **MD** line command. (Note that IBM ISPF has no counterpart to the SPFLite **MN** command.)

Note: Presently, there is no facility to convert a normal data or special lines into an **=xNOTE>** line. You also cannot directly convert an **xNOTE** into a standard **NOTE**. However, you can convert an **xNOTE** to a data line using **MD**, then use **MN** to change the data line to a standard **NOTE** line.

The **MN** line command can be issued on any normal line, special line or **NOTE** line. On the Top of Data and Bottom of Data lines, **MN** is not allowed. On excluded lines, the lines remain excluded after they are converted.

When a normal data line has a label or tag and is converted to a **=NOTE>** line, the label or tag is removed from the line; **NOTE** lines cannot be labeled or tagged.

It is legal to use **MN** or **MNN** on a line or block in which some or all of the lines are already **NOTE** lines. If this is done, these lines are simply ignored and remain **NOTE** lines.

To convert one or more ordinary data lines to **=NOTE>** lines:

1. Type **MN** in the line command area on the line that is to be converted. If you want to convert more than one line, type an 'n' value greater than 1 after the **MN**.
2. Press Enter. The data lines are converted to **=NOTE>** lines.

To convert a block of data lines to **=NOTE>** lines:

1. Type **MNN** in the line command area of both the first and last lines to be converted. You can scroll (or use the **FIND** or **LOCATE** command) between the first **MNN** and the second **MNN**, if necessary.
2. The **LOCATE [NOT] NOTE** command may be used to find the **FIRST**, **LAST**, **PREV** or **NEXT** line which is (or is not) a **=NOTE>** line.
3. The **LOCATE [NOT] SPECIAL** command may be used to find the **FIRST**, **LAST**, **PREV** or **NEXT** line which is (or is not) a special line, such as is displayed by the **PROFILE** command.
4. Press Enter. The lines that contain the two **MNN** commands and all lines between them are converted to **=NOTE>** lines.

Example:

Here, a number of lines have the Make Note command **MN** placed on them:

MD_Test.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MD_Test.TXT

File Manager MD_Test.TXT | Command > Scroll > HALF

```
***** * Top of Data *****  
MN PROFILE TXT UNLOCKED, AUTOBKUP OFF, AUTOCAPS ON, AUTOSAVE OFF PROMPT  
MN CAPS OFF, CASE T, CHANGE CS, COLLATE ANSI, COLS OFF, EOL CRLF, FOLD OFF  
MN HEX OFF, HILITE FIND AUTO, LRECL 0, MARK ON, MINLEN 0, PAGE OFF  
MN PRESERVE ON, RECFM U, SCROLL HALF, SETUNDO 5, SOURCE ANSI, START FIRST  
MN STATE ON, TABS ON, XTABS 4  
=WORD> A-Z a-z 0-9  
=MARK>  
=MASK>  
=TABS>  
=COLS> -----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8-----+  
=BNDS> <  
000006 Data Line 1  
=NOTE> This is a Note line to be kept  
000007 This is a Note line to be changed to a Data line  
MN Data Line 2 - TO BE CHANGED TO A NOTE  
***** * Bottom of Data *****
```

Result: Notice that the lines that had MN line commands placed on them now have the =NOTE> marker, because they are now **NOTE** lines.

N - Insert New (Permanent) Blank Lines

Syntax

```
N[n]
```

Operands

- n** The number of permanent blank lines to insert. If you do not type a number, or if the number you type is 1, only one line is inserted.

Description

To insert one or more permanent lines in a file:

1. Type **N** in the line command area of the line that the inserted line is to follow. If you want to insert more than one line, type a number greater than 1 after the **N** command.
2. Press Enter.
3. The lines are inserted.

The lines are inserted as permanent blank lines, unlike the [T](#) command, which inserts the lines as temporary blank lines. Whether you use the lines created by **N** or not, they will remain as data lines within the edit file.

Blank lines created by **N** are zero-length lines, and do not utilize the contents of the **MASK** line, even if a non-blank **MASK** line is defined.

Key-Mapping the N command

Users of character-mode text editors usually create new blank lines by pressing the Enter key. While this is possible to do in SPFLite, it is more common for the 'normal' SPFLite Enter key to be mapped to the [\(NewLine\)](#) function. If you would like a key to easily create new blank lines, a suggested mapping is to put the line command **{N}** in the entry for Ctrl-Enter in the KeyMap. Ctrl-Enter is easy to type on most keyboards, and is a good compromise for this function.

NOTE - Insert NOTE or xNOTE Lines

Syntax

```
NOTE [n]  or
xNOTE [n]
```

Operands

n The number of new, blank **=NOTE>** or **xNOTE>** lines to be created. If **n** is omitted or specified as zero, one line is created.

Up to 99 such lines can be directly created with a single **NOTE** line command, or up to 9 for **xNOTE**, unless the sequence number field has been made wider than 6 columns. If the sequence field has a width of 5, up to 9 **=NOTE>** lines can be created, and only a single **xNOTE>** line. See note below.

See Width of Line Numbers in Options - Screen for information about adjusting the width of the sequence number area.

Description

The **NOTE** or **xNOTE** line command provides a quick way to create a number of blank **=NOTE>** lines. These lines can then be typed-over with any desired information.

You can create either simple **NOTE** lines or use an alphabetic prefix character from **A** to **Y** to create different types of notes, such as **CNOTE**, **MNOTE**, etc.

xNOTE lines are most likely be used by some macro command, such as one used in conjunction with a compiler, to insert diagnostic message lines into the text based on the compiler output, perhaps using different **xNOTE** types for various error severity levels. SPFLite does not provide such macros; you would have to write them yourself.

The keyword **ZNOTE** is reserved for use on the primary command line; you cannot enter **ZNOTE** into the sequence area as a line command.

Inserting additional NOTE lines

Because of the length of the **NOTE** or **xNOTE** line commands, there is limited room for specifying the **n** value if you need to insert many blank note lines. There are two methods for creating more note lines than can be specified by the **n** value:

- If more note lines are needed, a single **=NOTE>** or **xNOTE>** line can be made first, then the **R** line command can be used to repeat the note line as many times as desired.
- Say you need 25 new **XNOTE** lines, but the **n** value only allows up to 9 (or, perhaps there is no room for the **n** at all). If these lines were to appear after line 100 in your file, you could create them from the primary command line using a **LINE** primary command. This works because, internally, SPFLite treats the sequence number field as if it were 8 columns wide, even if not all 8 columns are displayed):

```
LINE XNOTE25 .100
```

See [*Working with NOTE and xNOTE Lines*](#) for more information.

O / OO - Overlay Lines

Syntax

```
O[n]  
OO / OO
```

Operands

n The number of lines to be overlaid. If you do not type a number, or if the number you type is 1, only one line is overlaid.

Description

The **O** (overlay) line command specifies the destination of data that is to be copied or moved by the [C](#) (copy) or [M](#) (move) line commands. The data that is copied or moved overlays **blanks** in an existing line of data. This allows you to rearrange a single-column list of items into multiple column, or tabular, format.

To overlay one or more lines:

1. Type either [M](#) or [C](#) in the line command area of the line that is to be moved or copied.
2. Type [O](#) in the line command area of the line that the moved or copied line is to **overlay**. You can type a number after the **O** line command to specify the number of times that the [M](#) or [C](#) line command is to be performed.
3. Press Enter.
4. The data being moved or copied overlays the specified line or lines.

To overlay a block of lines:

1. Type either [MM](#) or [CC](#) in the line command area of the first and last lines of a block of lines that is to be moved or copied. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first command and the second command, if necessary.
2. Type [OO](#) in the line command area of the first and last lines that the block of lines being moved or copied is to overlay. Again, you can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first [OO](#) and the second [OO](#), if necessary.
3. Press Enter.
4. The lines that contain the two [CC](#) or [MM](#) commands and all of the lines between them overlay the lines that contain the two [OO](#) commands and all of the lines between them.

Only blank characters in the lines specified with **O** or **OO** are overlaid with corresponding characters from the source lines. Characters that are not blank are not overlaid. The overlap affects only those characters within the current column boundaries.

The number of source and receiving lines need not be the same. If there are more receiving lines, the source lines are repeated until the receiving lines are gone. If there are more source lines than receiving lines, the extra source lines are ignored. The overlay operation involves only data lines. Special lines such as TABS, BNDS, and COLS are ignored as either source or receiving lines.

Following the operation, if the source lines were selected with [W/MM](#) line commands, then the source lines will be deleted if, and only if, all characters in the source lines were overlaid into blanks in the receiving lines, **or** the characters in the receiving lines were identical.

Application Note: Using Excluded Lines for Overlay Operations

When you are overlaying one block of lines with another block of lines, and the content of these lines is indented or of variable length, it can sometimes be difficult to decide exactly which lines the line commands should be placed on. A convenient way to 'line up' the lines you want to use is to temporarily exclude the data lines, or some nearby lines, in order to make more obvious which data lines are the ones you want.

Using Excluded Lines as Guides

You can use an excluded line as a "marker" to identify the beginning or ending of a region of lines you want to move, copy or overlay. By putting an X line command where the ends of the region of lines are, you can see if you placed it correctly by visually inspecting the data lines that are next to the dashed line of the excluded-line placeholder. If it appears you misplaced this line, just use an S command to unexclude it, and try again. Once you have your area defined as you like, use the **MM**, **CC**, **OO** or **ORR** command as needed. Example:

```
***** ***** Top of Data *****
000001 other data lines
000002 other data lines
x
000004 A11
000005 A22
000006 A33
x
000008 other data lines
x other data lines
000010 B11
000011 B22
000012 B33
x
000014 other data lines
***** ***** Bottom of Data *****

Edit L 000013 Lines: 14 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0018
```

Notice how the excluded line placeholders act as visual guides, helping you to see exactly which lines are needed for the Overlay operation. Here, we want to use lines 4-6 and 10-12.

```
***** ***** Top of Data *****
000001 other data lines
000002 other data lines
-----
000004 A11 < 000001 >
000005 A22 < 000001 >
000006 A33 < 000001 >
-----
000008 other data lines < 000001 >
-----
000010 B11 < 000001 >
000011 B22 < 000001 >
000012 B33 < 000001 >
-----
000014 other data lines < 000001 >
***** ***** Bottom of Data *****

Edit L 000004 C 1 Lines: 14 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0003
```

Now, it is very obvious where to place the **OO** and **MM** line commands:

File Manager Overlay.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001 other data lines
000002 other data lines
----- < 000001 > -----
00 A11
000005 A22
00 A33
----- < 000001 > -----
000008 other data lines
----- < 000001 > -----
mm B11
000011 B22
mm B33
----- < 000001 > -----
000014 other data lines
***** **** Bottom of Data ****
```

Edit L 000012 Lines: 14 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

The Overlay operation is complete:

File Manager Overlay.TXT | Command > Scroll > HALF

```
***** **** Top of Data ****
000001 other data lines
000002 other data lines
----- < 000001 > -----
000004 A11 B11
000005 A22 B22
000006 A33 B33
----- < 000001 > -----
000008 other data lines
----- < 000002 > -----
000011 other data lines
***** **** Bottom of Data ****
```

Edit * L 000004 C 1 Lines: 11 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

Finally, **RESET** the display so the rest of the file becomes visible again.

```
*****
000001 other data lines
000002 other data lines
000004 A11 B11
000005 A22 B22
000006 A33 B33
000008 other data lines
000011 other data lines
*****
```

Using Excluded Line as the Source and/or Target of an Overlay Operation

When you have a large number of lines involved in an Overlay operation, it may be convenient to exclude the two blocks of data lines involved, so that you can confirm you have the correct number of lines in each part. Normally, overlays involving large blocks of lines are done with an equal number of lines in both the source block and the target block. Making the lines excluded is one way you can confirm the line counts. Once that is done, you can directly overly the block of lines by using non-block line commands **C**, **M**, **O** and **OR**. That is because the excluded-line placeholder represents **every** line in that excluded range. Example:

```
*****
000001 other data lines
000002 other data lines
000003
XX A11
000005 A22
XX A33
000007
000008 other data lines
000009 other data lines
X B11
X B22
X B33
000013 other data lines
000014 other data lines
*****
```

There are now two excluded ranges of lines. Because each range shows **< 000003 >** we are certain they are the same size. We now overlay the second range (lines 10-12) on to the first range (lines 4-6).

Because an excluded-line range is treated as a single entity, you do not use block commands OO or MM, but rather use line command O and M to move the entire source range (10-12) on top of the entire target range (4-6):

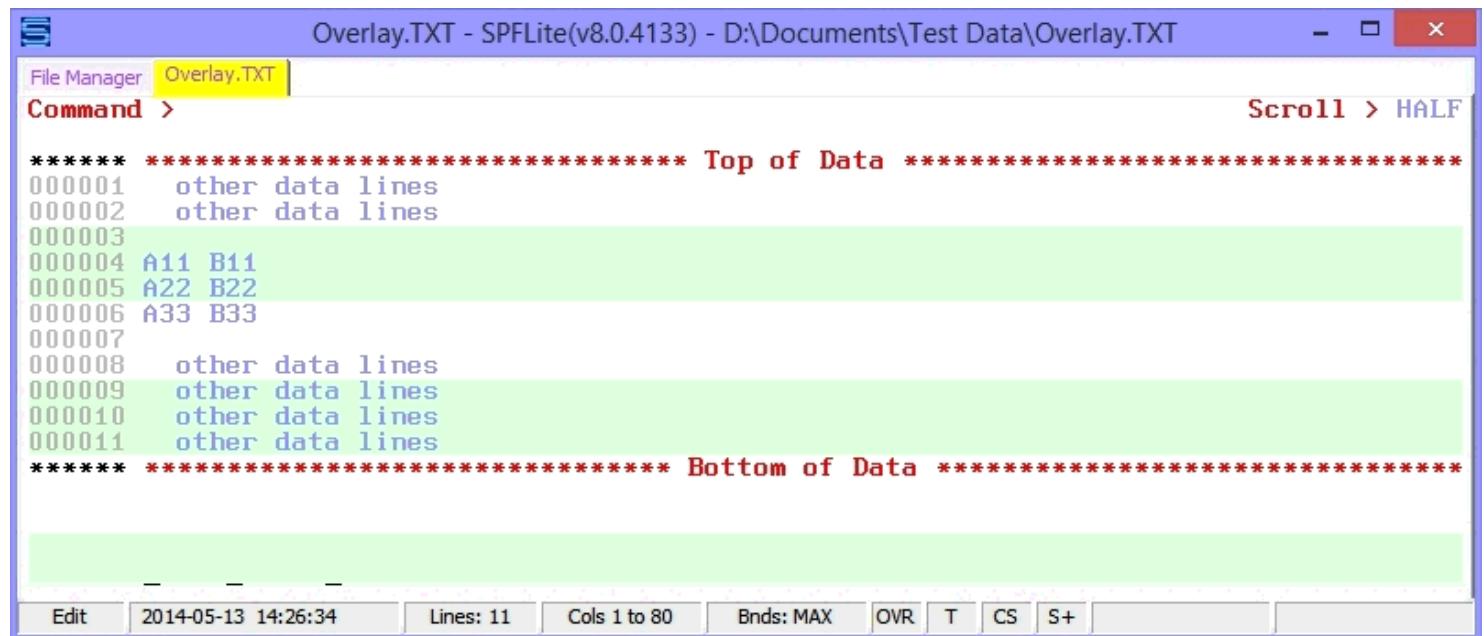
```
***** **** Top of Data ****
000001 other data lines
000002 other data lines
000003
0 ----- < 000003 > -----
000007
000008 other data lines
000009 other data lines
M ----- < 000003 > -----
000013 other data lines
000014 other data lines
***** **** Bottom of Data ****
```

Because both the source range and the target range were excluded, the target range remains excluded after the Overlay operation is completed:

```
***** **** Top of Data ****
000001 other data lines
000002 other data lines
000003
----- < 000003 > -----
000007
000008 other data lines
000009 other data lines
000010 other data lines
000011 other data lines
***** **** Bottom of Data ****
```

Finally, after unexcluding the 3 remaining excluded lines with a **RESET** command, the complete file is now visible.

The same result would have occurred if you had put an **S** line command (Show) on the excluded line placeholder that appears after line 3.



OR / ORR - Overlay-Replace Lines

Syntax

OR [n]	ORR is the block form of this command.
ORR / ORR	

Operands

n	The number of lines to be overlay-replaced. If you do not type a number, or if the number you type is 1, only one line is overlay-replaced.
---	---

Description

The Overlay-Replace line command operates similar to the standard Overlay line command, with a slight difference. Overlay-Replace is used to unconditionally overlay data from the 'sending' lines to the 'receiving' lines.

The standard Overlay line command takes a set of one or more 'sending' lines and overlays them on top of a set of one or more 'receiving' lines. Sending lines are marked by **C/CC** or **M/MM** line commands, and receiving lines are marked by **O/OO** or **O&/OO&** line commands. (The **&** versions are persistent variants of the standard **O** and **OO** commands).

In the standard Overlay, non-blank characters in the sending lines overlay blank characters in the receiving lines, on a character by character basis. A 'benign overlay' occurs if a non-blank sending character position is matched to a non-blank receiving character when the two character values are identical. When the two relative positions are not blank and are not identical, an overlay mismatch occurs. For sending lines marked with **C/CC**, the mismatch is ignored for that character in that column, and the receiving column is unchanged. For sending lines marked with **M/MM**, the mismatch is ignored for the character in that column, an error message is displayed, and the mismatch prevents the lines marked with **M/MM** from being deleted afterwards.

For Overlay-Replace command, these semantics are modified as follows.

The receiving lines for an Overlay-Replace are marked with the line command **OR**, or with a block command of **ORR** or **OROR**. For an Overlay-Replace command, the overlay mismatch condition that can occur for a standard Overlay command does not take place. There is no such thing as a mismatch in this command, and no mismatch error messages are displayed.

Instead, all non-blank characters from the sending lines are superimposed on the receiving lines, without regard to the prior contents of the receiving lines. Where a sending line has blank character positions, the corresponding character positions in the receiving lines are left unchanged. Because mismatch conditions can never occur, sending lines marked with **M/MM** are always deleted afterwards for the **OR** command.

O/OO Examples

Before:

```
O 0001 A CD
C 0002 ABX
```

After: (no message issued)

```
000001 ABCD
```

000002 ABX

Before:

O 0001 A CD
M 0002 ABX

After: (message: **MOVE data not deleted – all data not overlaid**, line 2 not deleted)

000001 ABCD
000002 ABX

OR/ORR Examples

Before:

OR 001 A CD
C 0002 ABX

After: (no message issued)

000001 ABXD
000002 ABX

Before:

OR 001 A CD
M 0002 ABX

After: (no message issued, line 2 is gone)

000001 ABXD **because line 2 is moved over line 1**

See [O / OO - Overlay Lines](#) for an **Application Note** regarding use of excluded lines when performing an overlay.

PL / PLL - Pad Lines to Length

Syntax

PL [n]	PLL is the block form of this command.
PLL[n] / PLL	

Operands

n	A number that tells the editor the minimum desired length of each line in the range. Shorter lines will be lengthened (by adding spaces). Longer lines will be untouched. If n is not specified, a value of 1 is assumed.
----------	--

Description

The **PL** command is used to force one or more lines to have an explicit minimum length. Its sole purpose is to selectively add trailing blanks; no other data changes take place. If a line is already equal or longer than the number specified, it is untouched.

The Pad to Length line command **PL** will accept a special modifier of **/** or ****. When used in this way, a command of **PL/** will pad all following lines to a minimum length of 1; **PL** will do the same to preceding lines.

Placing **PL/** on line 1 of a file can be used to ensure that all lines in the file have a minimum line length of 1. That is, it is a quick way to ensure there are no zero-length lines in the file.

R / RR - Repeat Lines

Syntax

```
R[n]  
RR[n] / RR
```

Operands

n The number of lines to be repeated. If you do not type a number, or if the number you type is 1, only the line on which you type **R** is repeated

Description

To repeat one or more lines:

1. Type **R** in the line command area of the line that is to be repeated. If you want to repeat the line more than once, type a number that is greater than 1 immediately after the **R** command.
2. Press Enter.
3. The editor inserts a duplicate copy or copies of the line immediately after the line that contains the **R** command.

To repeat a block of lines:

1. Type **RR** in the line command area of both the first and last lines to be repeated. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **RR** and the second **RR**, if necessary.
2. Press Enter.
3. The lines that contain the two **RR** commands and all of the lines between them are repeated immediately after the line that contains the second **RR** command.

Using the R command with the Forward and Backward Modifiers

If the **R** command is specified as **R/** or **R** it means all lines (forward or backward) are repeated as a block, one time.

S / SS - Show lines

Syntax

```
S [n]  
SS / ss
```

Operands

n The number of lines to be unexcluded. If you do not type a number, or if the number you type is 1, only the line on which you type **S** is unexcluded.

Description

To re-display one or more excluded lines:

1. Type **S** in the line command area of the line that is to be re-displayed. If you want to re-display more than one line, type a number that is greater than 1 immediately after the **S** command.
2. Press Enter.
3. The editor will re-display the specified lines.

To re-display a block of lines:

1. Type **SS** in the line command area of both the first and last lines to be re-displayed. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **SS** and the second **SS**, if necessary. Note: it is permissible for the specified range to include multiple non-contiguous excluded ranges.
2. Press Enter.
3. The lines that contain the two **SS** commands and all of the lines between them are re-displayed.

Note: Also see the [E](#) and [L](#) line commands for alternative re-display commands. Note also that the legacy ISPF line command **S** is now performed in SPFLite under the new line command name of [SI](#) (Show Indentation).

SC / SCC - Sentence-Case Lines

Syntax

SC [n]	SCC is the block form of this command.
SCC / SCC	

Operands

n	The number of lines to be converted to Sentence Case. If you do not type a number, or if the number you type is 1, only the line on which you type SC is sentence-cased.
---	---

Description

To convert characters on one or more lines to Sentence Case:

1. Type **SC** in the line command area of the line that contains the characters that you want to convert to Sentence case. To convert characters on lines following this one, type a number greater than 1 after the **SC** command.
2. Press Enter.
3. The characters on the specified line(s) are converted to Sentence Case.

To convert characters in a block of lines to Sentence Case:

1. Type **SCC** in the line command area of both the first and last lines that contain characters that are to be converted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **SCC** and the second **SCC**, if necessary.
2. Press Enter
3. The characters on the specified lines that contain the two **SCC** commands and in all of the lines between them are converted to Sentence Case.

Sentence Case description

Sentence Case processing is performed by first converting all the text to lower case. Then the first letter of the first word of each sentence is Capitalized.

Note:

- The first word of the first line processed is considered to be the start of a sentence.
- The last word of a sentence is considered to be any word ending with a period (.), Exclamation point (!), or a Question mark (?).

SI - Show Indentation

Syntax

```
SI [n]
```

Operands

n The number of lines to be redisplayed. If you do not type a number, or if the number you type is 1, only one line is redisplayed

Description

The **SI** (Show Indentation) line command causes one or more lines in a block of excluded lines to be redisplayed. The redisplayed lines have the leftmost indentation levels; they contain the fewest leading blanks.

Note that **SI** is the new name for the legacy ISPF line command of **S**. The SPFLite line command **S** now performs a different function. SPFLite has chosen to deviate from the ISPF standard here because the legacy Show Indentation feature is rarely used, and the SPFLite S/SS command performs a more important and useful operation.

If there are more than 2 excluded lines, and you do not type a number or if the number you type is 1, only one line is redisplayed.

Note: If you enter an SI line command to display all but one line of an excluded block, then that line is also displayed. This could result in more lines being displayed than the number you requested. For example, if five lines are excluded in a block, an SI4 command causes all five lines to be displayed.

To redisplay a line or lines of a block of excluded lines:

1. Type SI in the line command field next to the dashed line that shows where a line or lines has been excluded. The message in the dashed line tells you how many lines are excluded. If you want to redisplay more than one line, type a number greater than 1 after the SI command. If you type SI3, for example, the three lines with the leftmost indentation level are displayed again. If more than three lines exist at this indentation level, only the first three are displayed.
2. Press Enter. The line or lines with the fewest leading blanks are redisplayed.

T / TT - Select Text Lines

Syntax

```
T[n]
TT / TT
```

Operands

n The number of lines to be selected. If you do not type a number, or if the number you type is 1, only the line on which you type T is selected. The **n** value can be a special modifier / or \.

Description

The Text Selection line command **T/TT** may be used to select large amounts of text for use with keyboard primitive commands that need such defined areas of text, such as [\(Pen/Green\)](#). This can be useful in cases where the area that needs to be selected is so large that mouse selection or the use of shift/arrow keys is not possible or practical.

When more than one line of text is selected, a rectangular region is defined, in which the column range is column 1 through the right-most column of the longest line in the **TT** block.

For this reason, it is not possible to select a range of lines in a **TT** block if every line is of length zero, because there is literally no text to select.

When more than one line of text is selected, and the lines are of differing lengths, and you use a keyboard function like [\(Copy\)](#) to place the text into the clipboard, any lines that are shorter than the longest line will be padded with blanks, so that every line placed into the clipboard will be as long as the longest line in the **TT** block. **See example below.**

The **T/TT** command allows you to mark a large range of lines in a simple manner. Since you are marking only the vertical component of a marked block, note the following as to how the left/right borders of the marked area are chosen.

- If no **BOUNDS** are set (that is, if you see the default **Bnds: MAX** displayed on the status line), the left border will be column 1 and the right border will be set to the rightmost non-blank character in the line range.
- If **BOUNDS 'n' MAX** are set, the left border will be the left bounds value, and the right border will be set to the rightmost non-blank character in the line range.
- If **BOUNDS 'n1' 'n2'** are set, these bounds values will be used as the left and right borders of the selected text area.

To select the entire file, you can place a **TT/** command on line 1 and press Enter.

Only whole lines are selected by **T/TT**, even when non-default **BOUNDS** are in effect.

Limitations of T/TT line command and LINE primary command

The **T/TT** line command can be used by the **LINE** primary command, to apply **T/TT** to one or more lines.

However, when **T** is applied to more than one line, each individual **T** is applied to each line one at a time. The way that **T/TT** operates, only a single, contiguous block of highlighted data may exist as any given time. So, if you attempted to issue a command like **LINE T .11 .13 ALL**, only line 13 will be highlighted. If you try to manually highlight line 11 with a **T**, then line 12, then line 13, you will see how and why it works this way.

Usage

To select one or more lines:

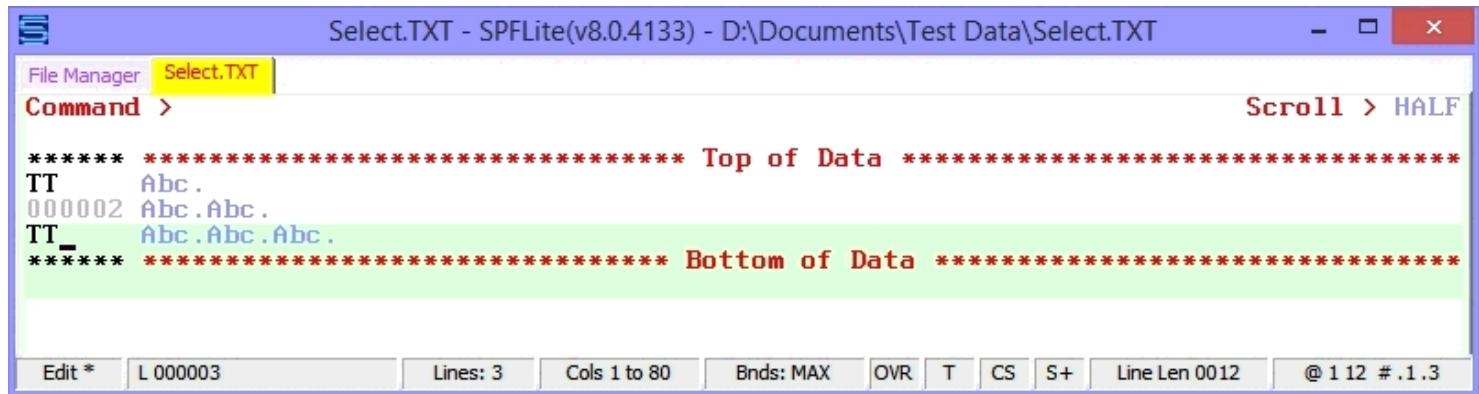
1. Type **T** in the line command area of the line that you want to select. If you want to select one or more lines that immediately follow this line, type a number greater than **1** immediately after the **T** command.
 2. Press Enter.
 3. The lines are selected.

To select a block of lines:

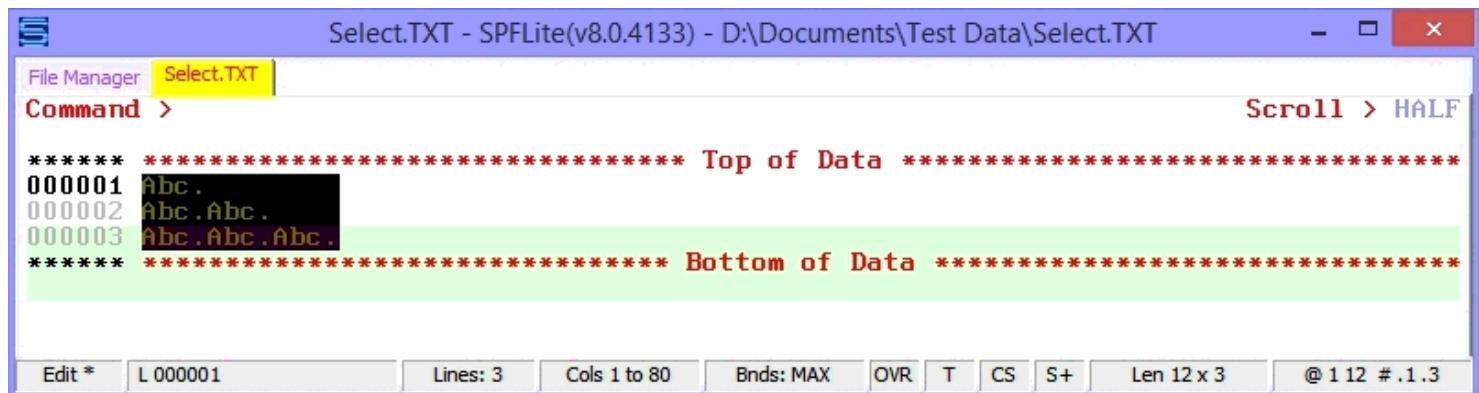
1. Type **TT** in the line command area of both the first and last lines that you want to select. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **TT** and the second **TT**, if necessary.
 2. Press Enter
 3. The lines that contain the two **TT** commands and all of the lines between them are selected and highlighted.

Example

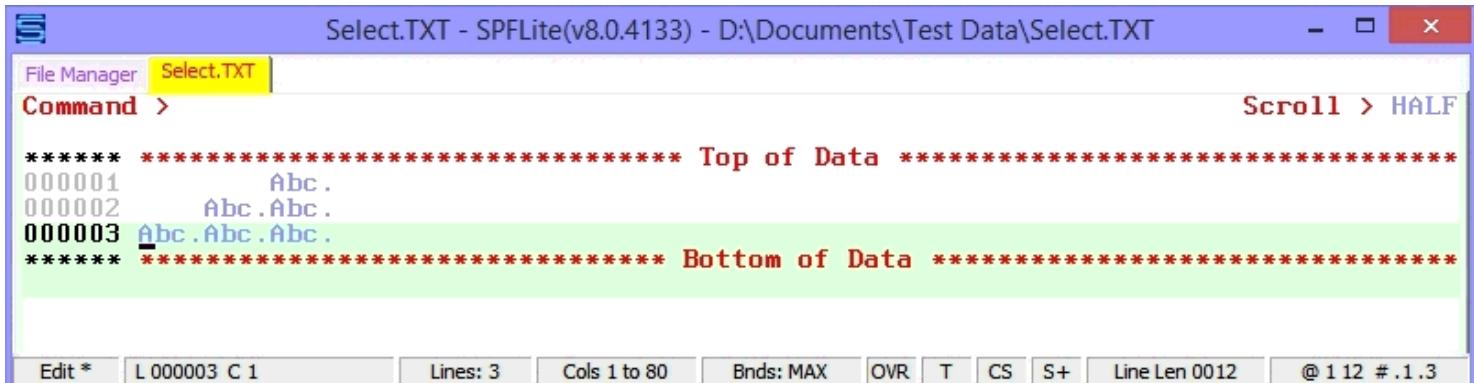
Three lines of differing lengths are selected with a TT block.



The last character of each line is the right-most period, but the TT block highlights these lines as a rectangular text block of length 12.



When a key mapped to the keyboard function ([JustifyR](#)) is used, the data is right-justified and made to align on column 12.



The screenshot shows the SPFLite editor interface with the file 'Select.TXT' open. The text content is as follows:

```
***** **** Top of Data ****
000001     Abc .
000002     Abc .Abc .
000003     Abc .Abc .Abc .
***** **** Bottom of Data ****
```

The text is right-justified, with each line starting at column 12. The status bar at the bottom shows the following information:

Edit * L 000003 C 1 Lines: 3 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0012 @ 1 12 # .1.3

TABS - Display TABS Line

Syntax

```
TABS
TAB
```

Operands

None

Description

When you type TABS in the line command area, =TABS> is displayed along with any previously defined tab positions. To remove the =TABS> line, use the [D](#) (delete) line command or the [RESET](#) primary command, or end the edit session. The =TABS> line is never saved as part of the data.

The tab definitions remain in effect, even if they are not displayed, until you change them. Tab definitions are retained, and are automatically used the next time you edit the same kind of data (based on the file extension). Note, the primary CAPS setting is saved along with TABS for each file type.

See "[Defining and Using TABS](#)" for more information on using Tabs.

TB / TBB - Text-Break Lines

Syntax

TB [n]	TBB is the block form of this command.
TBB / TBB	

n The number of permanent blank lines to be inserted between the broken lines. If you do not type a number, or if the number that you type is 1, the editor inserts only one permanent blank line.

The **n** value may be specified as **0** to request that no lines be inserted between the broken lines.

Description

Text Break will break one or more lines; each line is broken into two pieces based on where the cursor is positioned when the command is issued. This is similar to Text Split. Text Break will insert **permanent**, rather than **temporary**, blank lines. This means that any non-blank MASK line that might be defined will **not** be used.

To break one line:

1. Type **TB** in the line command area of the line you would like to be broken. If you want to insert more than one permanent blank line between the broken lines, type a number greater than **1** immediately after the **TB** command.

If you do not want any lines inserted between the broken lines, enter this command as **TB0**.

Note: When the command **TB0** is mapped to a key, it acts similar to how the Enter key does in a text editor like Notepad, by breaking apart a line at the point the key is pressed.

2. Move the cursor to the desired break point.
3. Press Enter
4. The line is broken at the cursor location and the requested number of permanent blank insert lines are added.

To break a block of lines:

1. Type **TBB** in the line command area of the first line you would like to be broken. If you want to insert more than one permanent blank line between the broken lines, type a number greater than **1** immediately after the **TBB** command.

If you do not want any lines inserted between the broken lines, enter this command as **TBB0**.

Type **TBB** in the line command area of the last line you would like to be broken. As usual, an **n** value need be specified on only one end of the the block, but if you specify it on both ends, the values must agree.

2. Move the cursor to the desired break point. Because the block form **TBB** will break each line in the block, you may put the cursor on any line within the **TBB** block, including excluded-line placeholders within the block.
3. Press Enter

4. Each line in the block is broken at the same cursor location and the requested number of permanent blank insert lines are added in between every pair of lines that are broken by **TBB**.

For ease of breaking an individual line, you may wish to map the **TB** command to a command key using [KEYMAP](#).

To rejoin lines, use the [TF](#) (Text Flow) line command or the [TM](#) (Text Margin) line command.

See [Word Processing Support](#) for more information on all SPFLite word processing support, and for an **Application Note** on splitting lines based on a search string.

The Join line commands **J/JJ** and **TJ/TJJ**, and the Glue line commands **G/GG** and **TG/TGG** can also be used to join lines together.

To insert temporary blank lines rather than permanent blank lines, see [TS - Text-Split a line](#).

See also the [SPLIT - Split Lines Using Find/Change Strings](#) primary command for more information.

TC / TCC - Title-Case Lines

Syntax

TC [n]	TCC is the block form of this command.
TCC / TCC	

Operands

n The number of lines to be converted to Title Case. If you do not type a number, or if the number you type is 1, only the line on which you type **TC** is title cased.

Description

To convert characters on one or more lines to Title Case:

1. Type **TC** in the line command area of the source code line that contains the characters that you want to convert to Title case. To convert characters on lines following this one, type a number greater than 1 after the **TC** command.
2. Press Enter
3. The characters on the specified line(s) are converted to Title Case.

To convert characters in a block of lines to Title Case:

1. Type **TCC** (or **TCTC**) in the line command area of both the first and last source code lines that contain characters that are to be converted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **TCC** and the second **TCC**, if necessary.
2. Press Enter.
3. The characters on the specified lines that contain the two **TCC** commands and in all of the lines between them are converted to Title Case.

Title Case Description

Title Case processing is performed by first converting all the text to lower case. Then the first letter of each word is Capitalized.

TF / TFF - Text-Flow a Paragraph

Syntax

TF [n]	TFF is the block form of this command.
TFF / TFF	

Operands

n	The column number to which the text should be flowed. The default is the panel width. If a number greater than the current maximum record length is specified, the maximum record length is used.
----------	---

Description

The Text Flow line command is used to reformat a paragraph of text to fit within a certain column range. A "paragraph" is a group of lines starting with the TF command and ending at the next blank line or the end of the file. The paragraph is made to "flow" by breaking it apart into "words" and reassembling the paragraph.

TF honors the existing text indent (normal or reverse) of the first and subsequent lines of the original paragraph.

Example:

```
First line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
```

would be re-flowed with an initial indent of 4 columns, and no indent on the remaining lines, whereas

```
First line of paragraph
    Subsequent line of paragraph
    Subsequent line of paragraph
    Subsequent line of paragraph
```

would be re-flowed with no initial indent, and an indent of 4 columns on the remaining lines.

The formatting is done this to retain existing paragraph structure, and is ISPF compliant.

To flow text within a single paragraph:

1. Type TF in the line command area of the line at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, type a number greater than 1 immediately after the TF command.
2. Press Enter.
3. The text is flowed from the beginning of that line to the end of the paragraph.

To text-flow multiple paragraphs with a single SPFLite interaction. This is an extension to IBM ISPF:

1. Type TFF in the line command area of the first line of the first paragraph at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, type a number greater than 1 immediately after the TFF command.

2. Type TFF in the line command area of the last line of the last paragraph at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, and did not specify this in step (1) above, you can type a number greater than 1 immediately after the second TFF command here. As with all block-mode commands, it is unnecessary but possible to specify a number both places, but if this is done, they must agree.
3. Press Enter.
4. The text is flowed from the beginning of the first line of the first paragraph to the end of last line of the last paragraph. The spacing between paragraphs is preserved.

Text Flow and BOUNDS

TF is sensitive to the current BOUNDS setting. This may be an issue when a paragraph already has text extending past the right bound. For example, if bounds are 1 80 and any lines in the TF line range are longer than 80, but the command TF80 is issued, this is rejected with a message,

TF right bound specified, but global BNDS are set. TF abandoned

whereas if a simple TF is used (without the **n** value) with non-default BOUNDS are in effect, no warning is issued - but no reformatting takes place either. Text Flow interacts with BOUNDS in an ISPF compliant manner. However, if this presents a problem for you, you can try using the Text Margin line command instead.

See [TM / TMM - Set Text Margin](#) and [Word Processing Support](#) for more information.

A note about using TF/TFF and TM/TMM to format text

The Text Flow command **TF/TFF** and the Text Margin command **TM/TMM** use a simplified definition of "words". A "word" is any string delimited by blanks or by the beginning or end of a line. It is not necessary to change the definition of the WORD string or the Normal Characters string for these commands to work correctly.

TG / TGG - Text Glue Lines

Syntax

TG[n] TGG is the block form of this command.
TGG / TGG

Operands

n The number of lines to be glued. If you do not type a number, or if the number you type is 1, the line on which the TG is entered will be joined to the line following it.

Description

TG/TGG is used to glue together one or more lines as 'logical' lines, with leading and trailing blanks trimmed off beforehand. The lines are glued into a single line by concatenating the trimmed original lines together left to right, in order. TG/TGG may be thought of as the Text Glue or Trimmed Glue operation.

By default, nothing is inserted between the glued lines; they are simply concatenated together. In effect, lines are glued together with an implied zero-length string between them.

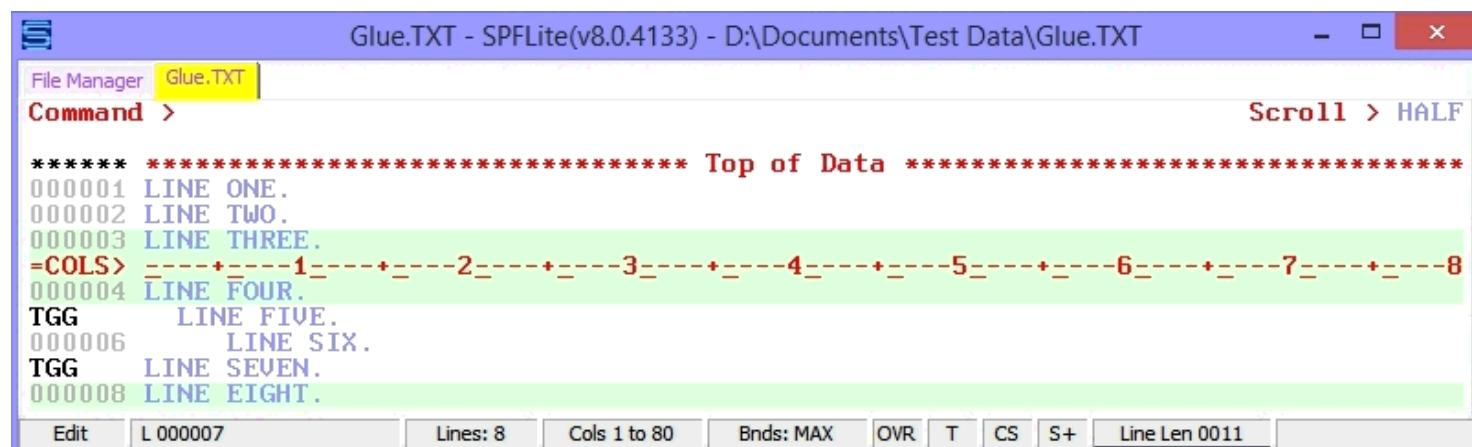
In some cases, it may be useful to specify a particular user-defined string to insert between lines. This is now possible using the **GLUEWITH** primary command. See the **GLUEWITH** command, and the example below, for more information. Note that **GLUEWITH** is a global setting, not associated with a particular file type.

Text Gluing of lines uses the following rules:

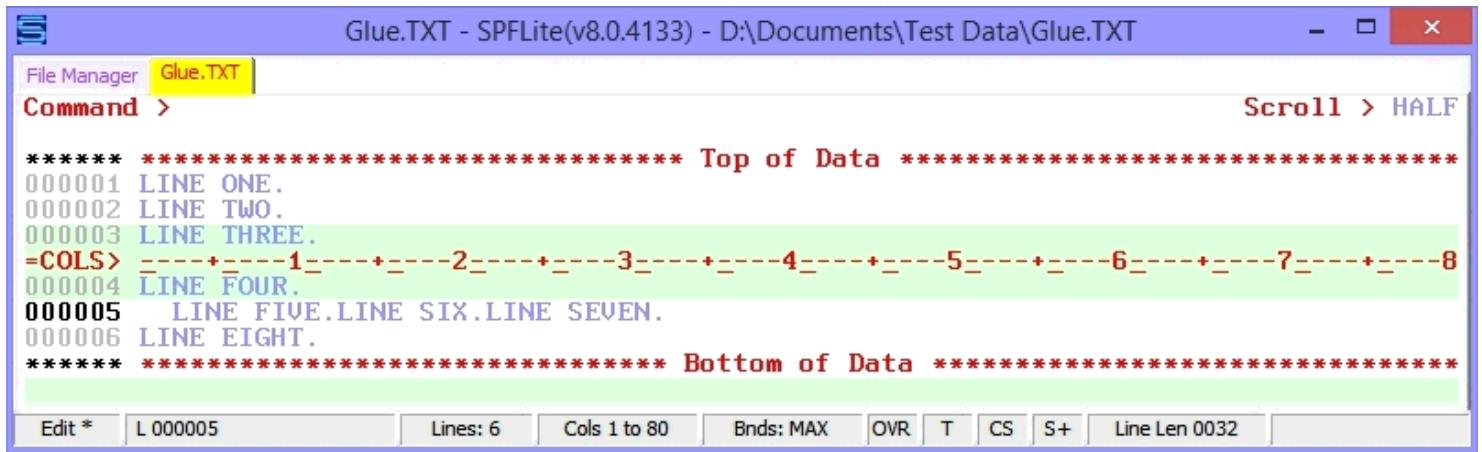
- Leading and trailing blanks are removed during the Text Glue process.
 - In the span of lines being glued, leading blanks from the first line, and trailing blanks from the last line, are not removed.
 - No space or other data is added following the last character of each line before appending the next line, unless **GLUEWITH** is in effect.

See also the [JOIN - Join lines Using Find/Change Strings](#) primary command for more information.

Example 1: Before Text Glue:



After Text Glue:



Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

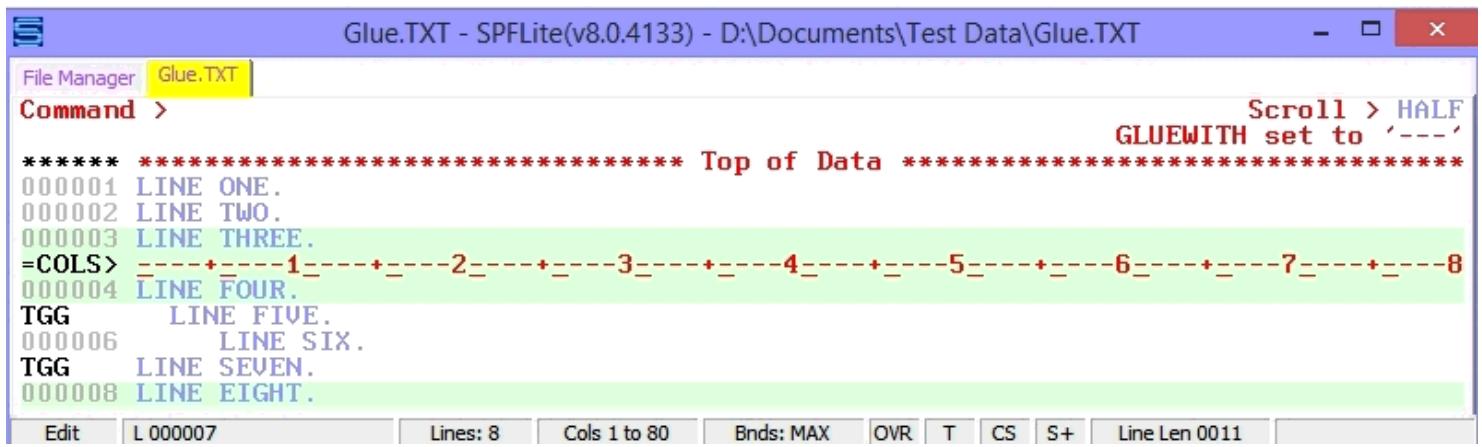
File Manager Glue.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+----1-----+----2-----+----3-----+----4-----+----5-----+----6-----+----7-----+----8
000004 LINE FOUR.
000005 LINE FIVE.LINE SIX.LINE SEVEN.
000006 LINE EIGHT.
***** **** Bottom of Data ****
```

Edit * L 000005 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0032

Example 2: Before Text Glue, **GLUEWITH '---'** in effect:



Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

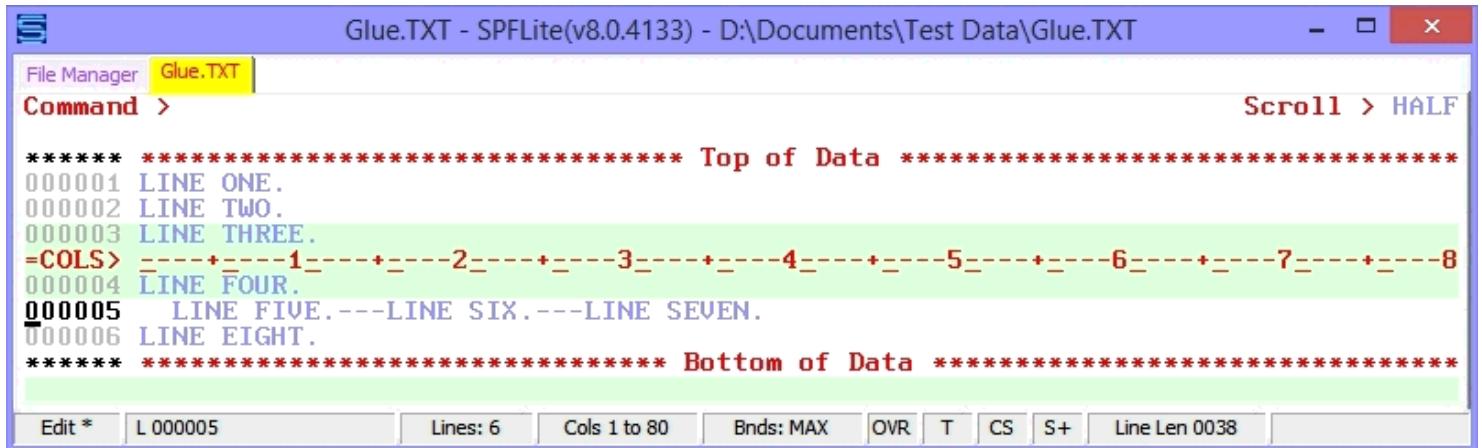
File Manager Glue.TXT

Command > Scroll > HALF
GLUEWITH set to '---'

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+----1-----+----2-----+----3-----+----4-----+----5-----+----6-----+----7-----+----8
000004 LINE FOUR.
TGG LINE FIVE.
000006 LINE SIX.
TGG LINE SEVEN.
000008 LINE EIGHT.
```

Edit L 000007 Lines: 8 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0011

After Text Glue:



Glue.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Glue.TXT

File Manager Glue.TXT

Command > Scroll > HALF

```
***** **** Top of Data ****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+----1-----+----2-----+----3-----+----4-----+----5-----+----6-----+----7-----+----8
000004 LINE FOUR.
000005 LINE FIVE.---LINE SIX.---LINE SEVEN.
000006 LINE EIGHT.
***** **** Bottom of Data ****
```

Edit * L 000005 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0038

TJ / TJJ - Text Join Lines

Syntax

TJ [n]	TJJ is the block form of this command.
TJJ / TJJ	

Operands

n The number of lines to be joined. If you do not type a number, or if the number you type is 1, the line on which the TJ is entered will be joined to the line following it.

Description

TJ/TJJ is used to join together one or more lines as 'logical' lines, with leading and trailing blanks trimmed off beforehand. The lines are joined into a single line by concatenating the trimmed original lines together left to right, in order, with a single blank character in between them. TJ/TJJ may be thought of as the Text Join or Trimmed Join operation.

A single blank character is inserted between the joined lines. To insert something else between the lines, it is necessary to use the Glue commands in conjunction with the GLUEWITH command.

Text Joining of lines uses the following rules:

- Leading and trailing blanks are removed from the original lines involved in the Text Join process.
- In the span of lines being joined, leading blanks from the first line, and trailing blanks from the last line, are not removed.
- A single space character is added following the last character of each line before appending the next line.
- The **GLUEWITH** string is not considered.

Note: Former Tritus SPF users may recognize the semantics of the SPFLite TJ command as essentially working the same way.

See also the [JOIN - Join lines Using Find/Change Strings](#) primary command for more information.

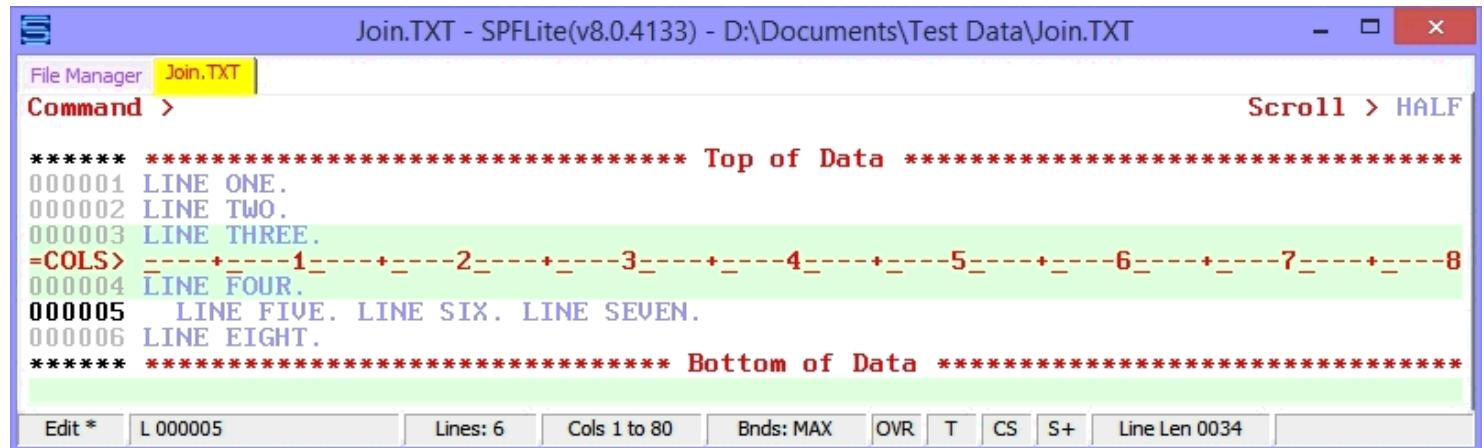
Example 1: Before Text Join:

```

Join.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Join.TXT
File Manager Join.TXT
Command > Scroll > HALF
***** * Top of Data *****
000001 LINE ONE.
000002 LINE TWO.
000003 LINE THREE.
=COLS> -----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----+-----8
000004 LINE FOUR.
TJJ LINE FIVE.
000006 LINE SIX.
TJJ LINE SEVEN.
000008 LINE EIGHT.

```

After Text Join:



TL / TLL - Truncate Lines to a Length

Syntax

TL [n]	TLL is the block form of this command.
TLL [n]	

Operands

n	A number that tells the editor the desired fixed length for each line in the range.
----------	---

Description

The TL command is used to force one or more lines to have an explicit fixed length. This action may involve deletion of non-blank data characters from the right-hand side of selected lines.

n is a fixed line size. It may be any non-zero value.

When the TL command imposes a fixed line length on one or more lines, it does this by truncating the right-hand side of all lines longer than **n** characters so that only **n** characters are left, as if the Erase EOF primitive keyboard operation were applied at cursor position **n+1**. Lines shorter than **n** characters are padded on the right with sufficient blanks to extend the line length to **n** characters.

If omitted on the command, **n** defaults to the length of the longest line of any line in the block. So for example, if lines 1 to 3 had line lengths of 5, 15 and 10, respectively, and TLL were placed on lines 1 and 3, all 3 lines would be given a length of 15.

This also means that a single TL command with no **n** value will effectively do nothing.

TM / TMM - Text Margin

Contents of Article

[Syntax](#)

[Operands](#)

[Description](#)

[Text Margin and BOUNDS](#)

[A note about using TF/TFF and TM/TMM to format text](#)

[Using TM to Emulate the TE command on ISPF](#)

Syntax

TM [n]	TMM is the block form of this command.
TMM / TMM	

Operands

n The column number to which the text should be flowed. The default is the panel width. If a number greater than the current maximum record length is specified, the maximum record length is used.

Description

The Text Margin line command is used to reformat a paragraph of text to fit within a certain column range. A "paragraph" is a group of lines starting with the TM command and ending at the next blank line or the end of the file. The paragraph is made to fit into the margin by breaking it apart into "words" and reassembling the paragraph.

Text Margin is similar to Text Flow (TF) except that the way it interacts with BOUNDS is different; see the discussion below.

TM honors the existing text indent (normal or reverse) of the first and subsequent lines of the original paragraph.

Example:

```
First line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
```

would be re-flowed with an initial indent of 4 columns, and no indent on the remaining lines, whereas

```
First line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
Subsequent line of paragraph
```

would be re-flowed with no initial indent, and an indent of 4 columns on the remaining lines.

The formatting is done this to retain existing paragraph structure, and is ISPF compliant.

The Text Margin command uses a simplified definition of "words", one that is not based upon the **Normal**

Characters for P'.' picture literals in the General Options dialog. Instead, a "word" is any string delimited by blanks or by the beginning or end of a line.

To flow text within a single paragraph:

1. Type TM in the line command area of the line at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, type a number greater than 1 immediately after the TM command.
2. Press Enter.
3. The text is flowed from the beginning of that line to the end of the paragraph.

You can use Text Margin to format multiple paragraphs with a single SPFLite interaction:

1. Type TMM in the line command area of the first line of the first paragraph at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, type a number greater than 1 immediately after the TMM command.
2. Type TMM in the line command area of the last line of the last paragraph at which you want the text to begin flowing. If you want to specify the rightmost column position for the restructured text, and did not specify this in step (1) above, you can type a number greater than 1 immediately after the second TMM command here. As with all block-mode commands, it is unnecessary but possible to specify a number both places, but if this is done, they must agree.
3. Press Enter.
4. The text is flowed from the beginning of the first line of the first paragraph to the end of last line of the last paragraph. The spacing between paragraphs is preserved.

Text Margin and BOUNDS

TM is affected by the current BOUNDS setting:

- When BOUNDS is defined as **1 bb** then TM without a **n** value is treated the same as if the command were specified as **TMbb**. When lines are reformatted into paragraphs, the right-hand margin of lines will be aligned so that they are as long as possible without the column margin exceeding a width of **bb** characters on any given line.
- When BOUNDS is defined as **1 MAX** then TM without a **n** value is treated the same as if the command were specified as **TMww**, where **ww** is the current screen width. When lines are reformatted into paragraphs, the right-hand margin of lines will be aligned so that they are as long as possible without the column margin exceeding a width of **ww** characters on any given line.
- TM **with a n value** respects the **n** value. When lines are reformatted into paragraphs, the right-hand margin of lines will be aligned so that they are as long as possible without the column margin exceeding a width of **n** characters on any given line. The current contents of the BOUNDS value is **ignored** when an **n** value is present on the TM command.

Regardless of any bounds values in effect, TM/TMM will place its results between column 1 and the right-most margin column as determined above, even if the current left bound of any BOUNDS command in effect would be incompatible with this understanding.

It is important to note that TM will **inquire** as to the state of BOUNDS at the time, but the results it produces are **not restricted** to the BOUNDS column range in effect at the time. In particular, regardless of the current BOUNDS setting, you will not get an error message similar to "**TF right bound specified, but global BNDS are set. TF abandoned**" that can sometimes occur with Text Flow.

See [TF / TFF - Text-Flow a Paragraph](#) and [Word Processing Support](#) for more information.

A note about using TF/TFF and TM/TMM to format text

The Text Flow command **TF/TFF** and the Text Margin command **TM/TMM** use a simplified definition of "words".

A "word" is any string delimited by blanks or by the beginning or end of a line. It is not necessary to change the definition of the WORD string or the Normal Characters string for these commands to work correctly.

Using TM to Emulate the TE command on ISPF

IBM ISPF contains the line command **TE** for Text Entry, a feature not supported in SPFLite. The IBM TE command was an attempt to perform a type of "word processing" on 3270-style terminals, which were not really well-suited for this task. On Windows, a comparable editing task would be to take free-form text (from an editor such as NotePad) that "wraps" from one line to the next, and convert it to discrete lines in the SPFLite editor. While there are probably several ways to accomplish this, here is a method that is likely to be the easiest and fastest way to go about doing it:

- In a character-mode editor such as NotePad, open an existing text file, or type the lines you need.
- If you wish, feel free to allow these lines of text to wrap from one line to the next without pressing Enter.
- Highlight (text-select) the desired lines, and press Ctrl-C to copy them into the Windows clipboard.
- In SPFLite, use an **A** line command, and the **PASTE** primary command, to paste the text data into your edit file.
- Use a **TMM** block to set the text margins of the lines you just pasted in, to some length (say, 80). This will reformat the block to a maximum length of 80 characters, splitting lines as needed and gluing others together.

Note: A convenient way to use the A line command and the PASTE primary command is to map Alt-V to the following key-mapping definition:

```
(saveCursor) (LineNo) [a] (home) [paste] (Enter) (restoreCursor)
```

Then, put your cursor on the line after which you want the text pasted in, and press Alt-V.

If you didn't want to go outside of SPFLite, you could simply create a bunch of temporary or permanent blank lines, enter your data, and use **TMM** to reformat them. This will accomplish basically the same thing, but will take more steps, because you will have to have several SPFLite interactions to create an "opening" in the file in which to put your initially-entered text before it gets reformatted.

TR / TRR - Trim Trailing Blanks

Syntax

TR [n]	TRR is the block form of this command.
TRR / TRR	

Operands

n	A number that tells the editor the desired minimum length of trimmed lines.
----------	---

Description

The TR command is used to trim trailing blanks from one or more lines. Its sole purpose is to selectively trim trailing blanks; no other data changes take place.

Here, **n** is a minimum line size. If omitted, **n** defaults to **0**, which is a legal **n** value. That is, if **n** is zero or omitted, the minimum resultant line length is zero, which, if applied to a "blank" line, will result in a line of length zero because the entire line's contents of blank characters has been deleted.

Lines which have no trailing blanks are left unchanged by the Trim command. When the **n** value is specified, and a given line already contains **n** or fewer characters, it is considered a "short line"; short lines are not trimmed any further, even if they contain trailing blanks.

Using the TR command with the Forward and Backward Modifiers

If the **TR** command is specified as **TR/** or **TR** it means all lines (forward or backward) are trimmed of all trailing blanks.

For example, if you wished to trim the entire file of trailing blanks, you can place a line command of **TR/** on line 1 and press Enter.

It is possible to map the TR command to a key in order to trim the entire file, by mapping **LINE TR/ .1** to an available key. A suggested key for this is Ctrl-Shift T.

TS - Text-Split a line

Syntax

```
TS [n]
```

Operands

n

The number of temporary blank lines to be inserted between the split lines. If you do not type a number, or if the number that you type is 1, the editor inserts only one blank line.

The **n** value may be specified as **0** to request that no lines be inserted between the split lines.

Description

To split a line:

1. Type **TS** in the line command area of the line you would like to split. If you want to insert more than one temporary blank line between the split lines, type a number greater than **1** immediately after the **TS** command.

If you do not want any lines inserted between the split lines, enter this command as **TS0**.

Note: When the command **TS0** is mapped to a key, it acts similar to how the Enter key does in a text editor like Notepad, by breaking apart a line at the point the key is pressed.

2. Move the cursor to the desired split point.
3. Press Enter.
4. The line is split at the cursor location and the requested number of temporary blank insert lines are added.

For ease of splitting an individual line, the **TS** command is often mapped to a key using [KEYMAP](#). A suggested key to map for this purpose is **Alt-S**.

To rejoin lines, use the [TF](#) (text flow) line command.

See [Word Processing Support](#) for more information on all SPFLite word processing support, and for an **Application Note** on splitting lines based on a search string.

The Join line commands **J/JJ** and **TJ/TJJ**, and the Glue line commands **G/GG** and **TG/TGG** can also be used to join lines together.

To insert permanent blank lines rather than temporary blank lines, see [TB / TBB - Text-Break Lines](#).

Relationship between TS, I and MASK Line Commands

The blank lines that are inserted at the point a line is split using the **TS** command are created in the same way that the **I** line command creates new lines. That means that the new lines are temporary, and will contain **.....** in the sequence area. You must manually type some data onto such temporary new lines to retain them. Otherwise, those new lines will disappear the next time you press Enter, the same way that temporary new lines are treated when the **I** line command creates new lines.

In particular, this means that if a non-default, non-blank **MASK** line has been defined, one or more copies of the **MASK** line will be inserted at the point the line is split via **TS**.

See also the [SPLIT - Split Lines Using Find/Change Strings](#) primary command for more information.

TU / TUU - Toggle User status of lines

Syntax

```
TU [n]  
TUU
```

Operands

n The number of lines whose User Line state is to be toggled. If specified as 0 or omitted, 1 is assumed.

Description

The **TU** command will toggle the User Line state of the lines within the range of the command.

Lines that formerly were User Lines (U lines) will become ordinary lines (V lines), and lines that formerly were ordinary lines (V lines) will become User Lines (U lines).

Lines that are U lines are marked by a | vertical bar in the "gap column" while ordinary lines (V lines) do not have this mark.

When the User Line state of a line is toggled, the presence or absence of the | vertical bar will be the opposite of what it was before.

It is possible to use the [LINE](#) primary command to apply **TU** to a range of lines.

It is also possible to map **TU** to a key to quickly toggle the User Line status of a single line.

Here is an example mapping to do that, which will keep the cursor on the current line when finished:

```
{TU} (enter) (up)
```

TX / TXX - Toggle Excluded status of lines

Syntax

```
TX [n]  
TXX
```

Operands

n The number of lines whose Excluded state is to be toggled. If specified as 0 or omitted, 1 is assumed.

Description

The **TX** command will toggle the Excluded state of the lines within the range of the command.

Lines that formerly were excluded will become unexcluded, and lines that formerly were unexcluded will become excluded.

It is possible to use the [LINE](#) primary command to apply **TX** to a range of lines.

It is also possible to map **TX** to a key to quickly toggle the Excluded status of a single line.

Here is an example mapping to do that, which will keep the cursor on the current line when finished:

```
{TX} (enter) (up)
```

U / UU - Mark User lines

Syntax

```
U[n]
UU / uu
```

Operands

n The number of lines to be marked as User lines. If specified as 0 or omitted, 1 is assumed.

Description

User lines are a way to specially identify lines for further processing. They provide a selection technique for other commands to identify which lines are (or are not) to be processed.

Conceptually, all data lines exist in one of two states: Either they are "U lines" or they are "V lines", where U lines ("User lines") are a set of one or more lines of special interest at some particular time, and V lines are everything else. User lines are marked with a | vertical bar in the "gap column", and V lines are unmarked. The unmarked V lines are the same, ordinary data lines you have always worked with.

A file will ordinarily consist entirely of V lines, which is the default state of a file. Other than having one or the other of these U/V states, there is no difference between U and V lines and the data lines you have always worked with.

Just as you can use the **X** and **NX** keywords on primary commands to select between excluded or non-excluded lines, you can use the **U** and **NU** keywords on primary commands to select between User or non-User lines.

Any primary command that accepts the **X|NX** option will accept the **U|NU** option.

The User line status is independent of other line characteristics. That means that any of these attributes may be applied to a line independently of each other:

- a line can be excluded, or not excluded
- a line can have a label, or not
- a line can have a tag, or not
- a line can be designated as a User line, or not

If you have set **STATE ON** for the file type you are editing, the User Line state of a file is persistent, so that the User/non-User state of the edit lines is retained when you close SPFLite and restart it later.

It is not an error to attempt to make a line into a User Line when it already is one. The request will be processed, and the state of the line will simply be unchanged.

To mark one or more lines as User lines:

1. Type **U** in the line command area of the source code line that you wish to be marked as a User line. To mark lines following this one, type a number greater than 1 after the **U** command.
2. Press Enter.

3. The specified lines will be marked as User lines.

To mark a block of lines as User lines:

1. Type **UU** in the line command area of both the first and last source code lines that contain the lines that are to be marked. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **UU** and the second **UU**, if necessary.
2. Press Enter.
3. The specified lines that contain the two **UU** commands and in all of the lines between them are marked as User lines.

Example

(before):

```
000010 line ten
UU      line eleven
UU      line twelve
000013 line thirteen
```

After **UU** processing:

```
000010 line ten
000011|line eleven
000012|line twelve
000013 line thirteen
```

Note that the selected lines now contain a | vertical bar in the "gap column", the space between the line number and the data area.

This is your visual indication that these lines are marked as User lines.

See [Working with User lines](#) for more information.

UC / UCC - Upper-Case Lines

Syntax

UC [n] UCC / UCC	UCC is the block form of this command.
-----------------------------------	--

Operands

n The number of lines to be converted to upper case. If you do not type a number, or if the number you type is 1, only the line on which you type UC is upper cased.

Description

To convert characters on one or more lines to upper case:

1. Type **UC** in the line command area of the source code line that contains the characters that you want to convert to upper case. To convert characters on lines following this one, type a number greater than 1 after the **UC** command.
2. Press Enter.
3. The characters on the specified line are converted to upper-case.

To convert characters in a block of lines to upper case:

1. Type **UCC** in the line command area of both the first and last source code lines that contain characters that are to be converted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **UCC** and the second **UCC**, if necessary.
2. Press Enter.
3. The characters on the specified lines that contain the two **UCC** commands and in all of the lines between them are converted to upper case.

V / VV - Revert User Line status

Syntax

```
V[n]  
VV / vv
```

Operands

n The number of lines whose User Line status is to revert back to that of ordinary lines (V lines). If specified as 0 or omitted, 1 is assumed.

Description

User lines are a way to specially identify lines for further processing. They provide a selection technique for other commands to identify which lines are (or are not) to be processed.

Conceptually, all data lines exist in one of two states: Either they are "U lines" or they are "V lines", where U lines ("User lines") are a set of one or more lines of special interest at some particular time, and V lines are everything else. User lines are marked with a | vertical bar in the "gap column", and V lines are unmarked. The unmarked V lines are the same, ordinary data lines you have always worked with.

A file will ordinarily consist entirely of V lines, which is the default state of a file. Other than having one or the other of these U/V states, there is no difference between U and V lines and the data lines you have always worked with.

Just as you can use the **X** and **NX** keywords on primary commands to select between excluded or non-excluded lines, you can use the **U** and **NU** keywords on primary commands to select between User or non-User lines.

Any primary command that accepts the **X|NX** option will accept the **U|NU** option.

The User line status is independent of other line characteristics. That means that any of these attributes may be applied to a line independently of each other:

- a line can be excluded, or not excluded
- a line can have a label, or not
- a line can have a tag, or not
- a line can be designated as a User line, or not

If you have set **STATE ON** for the file type you are editing, the User Line state of a file is persistent, so that the User/non-User state of the edit lines is retained when you close SPFLite and restart it later.

To **revert** a line means to change the state of a line from being a User Line to being an ordinary line (a V line).

It is not an error to attempt to revert a line that is already an ordinary line. The request will be processed, and the state of the line will simply be unchanged.

To revert one or more lines from being User lines to being ordinary (V) lines:

1. Type **V** in the line command area of the source code line that you wish to be reverted from a User line to an ordinary line. To revert lines following this one, type a number greater than 1 after the **V** command.

2. Press Enter.
3. The specified lines will revert to being ordinary (V) lines.

To revert a block of lines:

1. Type **VV** in the line command area of both the first and last source code lines that contain the lines that are to be un-marked. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **VV** and the second **VV**, if necessary.
2. Press Enter.
3. The specified lines that contain the two **VV** commands and in all of the lines between them are un-marked as User lines.

Example

(before):

```
000010 line ten
VV      |line eleven
VV      |line twelve
000013 line thirteen
```

After UU processing:

```
000010 line ten
000011 line eleven
000012 line twelve
000013 line thirteen
```

Note the selected lines, which had a | in the space between the line number and the actual text to indicate User status, have now been returned to normal status as ordinary lines (V lines).

W / WW - Swap Lines

Syntax

```
W[n]
WW / WW
```

Operands

n The number of lines to be swapped. If you do not type a number, or if the number you type is 1, only the line on which you type W is swapped. For the WW command, the *n* operand cannot be used.

Description

The W/WW line command is used to mark off one or more lines which are swapped with one or more lines marked with M or MM. W and WW are only valid for swapping lines between an M/MM block, and cannot be used for any other purpose.

See the description of M/MM for more information.

Note: See [Word Processing Support](#) for information about text swaps.

Example of using M/MM and W/WW to swap lines:

Before:

```
***** ***** Top of Data *****
MM LINE ONE.
MM LINE TWO.
000003 STUFF
000004 THINGS
WW LINE FIVE.
WW LINE SIX.
***** ***** Bottom of Data *****
```

After:



File Manager MoveSwap.TXT

Command >

Scroll > HALF

***** ***** Top of Data *****

000001 LINE FIVE.
000002 LINE SIX.
000003 STUFF
000004 THINGS
000005 LINE ONE.
000006 LINE TWO.

***** ***** Bottom of Data *****

Edit * L 000001

Lines: 6

Cols 1 to 80

Bnds: MAX

INS

T

CS

S+

Line Len 0010

WORD - Display Valid Word Characters

Syntax

```
WORD
```

Operands

None

Description

The **WORD** line command will insert a **WORD** line display into the edit file. This line will contain the current list of characters which are considered valid Word characters when SPFLite is searching for strings with a **FIND**, **CHANGE** or similar command, and a **WORD**, **PREFIX** or **SUFFIX** search option is used. The characters are used to determine the boundaries of these strings, as follows:

- A WORD string consists of WORD characters with a non-WORD character (delimiter) on both the left side and the right side
- A PREFIX string consists of WORD characters with a non-WORD character (delimiter) on the left side and a valid WORD character on the right side.
- A SUFFIX string consists of WORD characters with a non-WORD character (delimiter) on the right side and a valid WORD character on the left side.

Note also:

- A CHARS string disregards whether the string is or is not delimited by non-WORD characters
- A blank is always considered a non-WORD character
- The physical beginning and end of a line are always considered to be non-WORD delimiters, **as if** the line were preceded and followed by a blank character

Some languages extend the allowable characters which make up a 'word' to include special characters such as '_' (underscore). The WORD support allows you to customize the set of characters that are considered as valid characters to better support such languages.

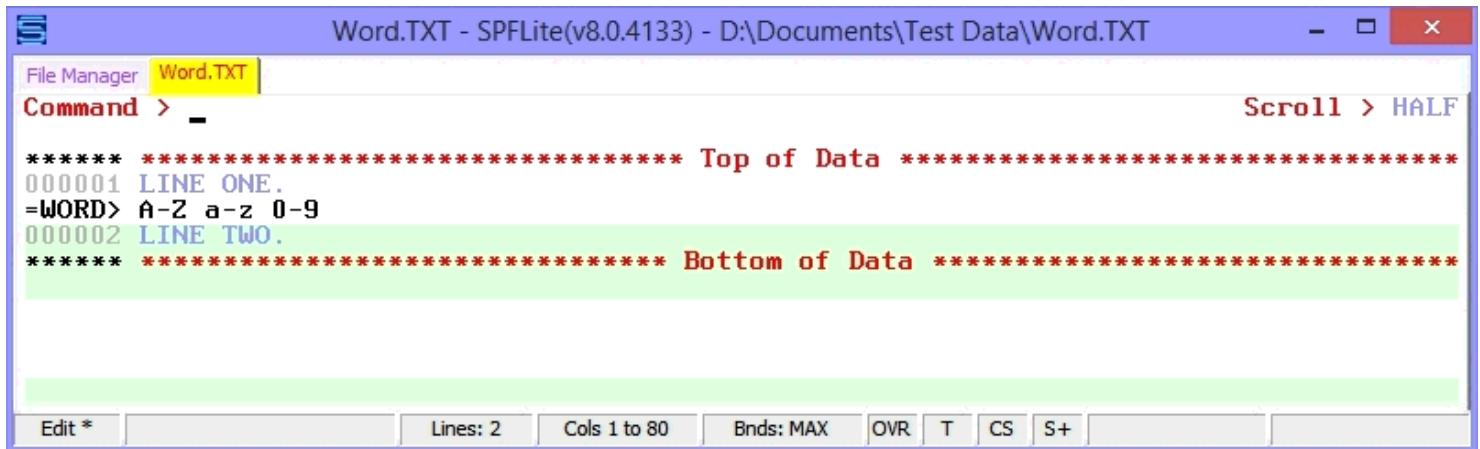
For example, if you are editing a "C" program file which allows underscores in names, the underscore is a letter-like character and not a delimiter. To successfully find "C" words in such files, you would **add** the underscore to the list of characters in the WORD line. Likewise, many IBM mainframe languages use \$ # and @ as letter characters.

Since this set of valid word characters will vary by language, SPFLite retains the WORD setting by file type in the PROFILE. It automatically selects and uses the WORD characters (whether the standard ones, or as modified by you) each time you edit a file of that type.

To display the word delimiters (=WORD>) line:

1. Type **WORD** in the line command area of any unflagged line.
2. Press Enter.
3. The word-delimiter line is displayed.

The delimiter line looks like this:



```
***** ***** Top of Data *****
000001 LINE ONE.
=WORD> A-Z a-z 0-9
000002 LINE TWO.
***** Bottom of Data *****
```

You may also enter the primary command PROFILE with no operands. One of the Profile lines displayed will be the =WORD> line.

1. Modify the characters on the **WORD** line to meet your needs. Full details of WORD syntax are available in [Working with Word and Delimiter Characters](#).
2. If you want the default set of characters restored, simply blank the entire line. You might want to do that if you made a mistake and want to start over with the standard WORD characters.
3. Press Enter.
4. The new word characters are now in effect.

To remove the WORD line from the panel:

1. You can either type **D** in the line command area that contains the =WORD> flag or type [RESET](#) on the Command line.
2. Press Enter.
3. The =WORD> line is removed from the display. Deleting the =WORD> line does not remove or change the WORD definition in any way; it simply stops displaying the line.

A note about using TF/TFF and TM/TMM to format text

The Text Flow command **TF/TFF** and the Text Margin command **TM/TMM** use a simplified definition of "words". For formatting purposes, a "word" is any string delimited by blanks or by the beginning or end of a line. It is not necessary to change the definition of the **WORD** string for these commands to work correctly.

X / XX - Exclude Lines

Syntax

```
X[n]  
XX / XX
```

Operands

n The number of lines to be excluded. If you do not type a number, or if the number you type is 1, only the line on which you type X is excluded.

Description

To exclude one or more lines:

1. Type X in the line command area of the line that you want to exclude. If you want to exclude one or more lines that immediately follow this line, type a number greater than 1 immediately after the X command.
2. Press Enter
3. The lines are excluded.

To exclude a block of lines:

1. Type XX in the line command area of both the first and last lines that you want to exclude. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first XX and the second XX, if necessary.
2. Press Enter
3. The lines that contain the two XX commands and all of the lines between them are excluded.

See [Working with Excluded Lines](#) for more information.

(Column Shift Left

Syntax

```
(  
(n  
(()  
(()n
```

Operands

n	A number that tells the editor how many positions to shift. If you omit this operand, the default is 2.
----------	---

Description

The `(` Column Shift Left line command moves characters on a line to the left without altering their relative spacing. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift one line toward the left side of your display:

1. Type `(` in the line command area of the line to be shifted. Beside the command, type a number other than 2 if you want to shift the line other than 2 columns. For example, to shift left 4 columns `(4` would be used.

When the **n** value is omitted, SPFLite uses the the "Default number of columns for data shifts" defined in the General Options dialog. The installation default for this value is 2, for compatibility with IBM ISPF. You can change this to any desired positive value. For example, if you set the value to 4, then when you used the `(` line command with no **n** value, it would shift 4 columns instead of 2.

2. Press Enter.
3. The line data is shifted the requested number of positions. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift a block of lines toward the left side of your display:

1. Type `((` in the line command area of the first line to be shifted. Beside the command, type a number other than 2 if you want to shift the block of lines other than 2 columns. For example, to shift left 4 columns `((4` would be used
2. Type `((` in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first `((` and the second `((` if necessary.
3. Press Enter.
4. The lines that contain the two `((` commands and all of the lines between them are column shifted to the left. Characters shifted past the current [BOUNDS](#) setting are deleted.

The [BOUNDS](#) setting limits column shifting. If you shift columns beyond the current [BOUNDS](#) setting, the editor deletes the text beyond the [BOUNDS](#) without displaying a warning message.

) Column Shift Right

Syntax

```
)  
) n  
))  
)) n
```

Operands

n A number that tells the editor how many positions to shift. If you omit this operand, the default is 2.

Description

The `)` Column Shift Right line command moves characters on a line to the right without altering their relative spacing. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift one line toward the right side of your display:

1. Type `)` in the line command area of the line to be shifted. Beside the command, type a number other than 2 if you want to shift the line other than 2 columns. For example, to shift right 4 columns `) 4` would be used.

When the **n** value is omitted, SPFLite uses the the "Default number of columns for data shifts" defined in the General Options dialog. The installation default for this value is 2, for compatibility with IBM ISPF. You can change this to any desired positive value. For example, if you set the value to 4, then when you used the `)` line command with no **n** value, it would shift 4 columns instead of 2

2. Press Enter
3. The line data is shifted the requested number of positions. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift a block of lines toward the right side of your display:

1. Type `))` in the line command area of the first line to be shifted. Beside the command, type a number other than 2 if you want to shift the block of lines other than 2 columns. For example, to shift right 4 columns `)) 4` would be used.
2. Type `))` in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first `))` and the second `))` if necessary.
3. Press Enter.
4. The lines that contain the two `))` commands and all of the lines between them are column shifted to the right. Characters shifted past the current [BOUNDS](#) setting are deleted.

The [BOUNDS](#) setting limits column shifting. If you shift columns beyond the current [BOUNDS](#) setting, the editor deletes the text beyond the [BOUNDS](#) without displaying a warning message.

< Data Shift Left

Syntax

```
<
<n
<<
<<n
```

Operands

- n** A number that tells the editor how many positions to shift. If you omit this operand, the default is 2.

Description

The `<` Data Shift Left line command moves characters on a line to the left without altering their relative spacing. Characters will NOT be shifted past the current [BOUNDS](#) setting. If necessary, the number of shifted columns will be reduced.

Note: See [Shifting Data](#) for more information on the exact shifting process used.

To Data Shift one line toward the left side of your display:

1. Type `<` in the line command area of the line to be shifted. Beside the command, type a number other than 2 if you want to shift the line other than 2 columns. For example, to shift left 4 columns `<4` would be used.

When the **n** value is omitted, SPFLite uses the the "Default number of columns for data shifts" defined in the General Options dialog. The installation default for this value is 2, for compatibility with IBM ISPF. You can change this to any desired positive value. For example, if you set the value to 4, then when you used the `<` line command with no **n** value, it would shift 4 columns instead of 2.

2. Press Enter.
3. The line data is shifted the requested number of positions. The shift is non-destructive. No characters will be dropped at the [Bounds](#) locations.

To Data Shift a block of lines toward the left side of your display:

1. Type `<<` in the line command area of the first line to be shifted. Beside the command, type a number other than 2 if you want to shift the block of lines other than 2 columns. For example, to shift left 4 columns `<<4` would be used.
2. Type `<<` in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first `<<` and the second `<<` if necessary.
3. Press Enter
4. The lines that contain the two `<<` commands and all of the lines between them are Data shifted to the left. The shift is non-destructive. No characters will be dropped at the [Bounds](#) locations.

The [BOUNDS](#) setting limits Data shifting. Data shifting is particularly oriented to programming source. See [Shifting Data](#) for more information.

> Data Shift Right

Syntax

```
>
>n
>>
>>n
```

Operands

- n** A number that tells the editor how many positions to shift. If you omit this operand, the default is 2.

Description

The **>** Data Shift Right line command moves characters on a line to the right without altering their relative spacing. Characters will NOT be shifted past the current [BOUNDS](#) setting. If necessary, the number of shifted columns will be reduced.

Note: See [Shifting Data](#) for more information on the exact shifting process used.

To Data Shift one line toward the right side of your display:

1. Type **>** in the line command area of the line to be shifted. Beside the command, type a number other than 2 if you want to shift the line other than 2 columns. For example, to shift right 4 columns **>4** would be used.

When the **n** value is omitted, SPFLite uses the "Default number of columns for data shifts" defined in the General Options dialog. The installation default for this value is 2, for compatibility with IBM ISPF. You can change this to any desired positive value. For example, if you set the value to 4, then when you used the **>** line command with no **n** value, it would shift 4 columns instead of 2.

2. Press Enter
3. The line data is shifted the requested number of positions. Characters will NOT be shifted past the current [BOUNDS](#) setting. If necessary, the number of shifted columns will be reduced.

To Data-shift a block of lines toward the right side of your display:

1. Type **>>** in the line command area of the first line to be shifted. Beside the command, type a number other than 2 if you want to shift the block of lines other than 2 columns. For example, to shift right 4 columns **>>4** would be used.
2. Type **>>** in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **>>** and the second **>>** if necessary.
3. Press Enter
4. The lines that contain the two **>>** commands and all of the lines between them are Data-shifted to the right. Characters will NOT be shifted past the current [BOUNDS](#) setting, if necessary, the number of shifted columns will be reduced.

The [BOUNDS](#) setting limits Data shifting. Data-shifting is particularly oriented to programming source. See [Shifting Data](#) for more information.

[Indent Shift Left

Syntax

```
[  
[n  
[[  
[[n
```

Operands

n A number that tells the editor how many *indent increments* to shift. If you omit this operand, the default is 1.

Description

The `[` Indent Shift Left line command moves characters on a line to the left without altering their relative spacing. Characters shifted past the current [BOUNDS](#) setting are deleted.

The command shifts in **indent increments**, where the indent increment is specified by the Options -> General value specified by the "Default # cols for Data shift" entry. This value is treated as the size of each indent level. If an **n** value is specified, the number of **columns** shifted will be **n** times the Options setting. For example, if the Options setting is 3, and a command `[[4` is entered, the lines will be shifted left **12** columns.

For comparison, an Indent Shift respects column boundaries and manages data the same way that a Column Shift does. An Indent Shift may thus be thought of as a specialized type of Column Shift.

To Indent Shift one line toward the left side of your display:

1. Type `[` in the line command area of the line to be shifted. Beside the command, type a number other than 1 if you want to shift the line other than 1 indent level. For example, to shift left 2 indent levels `[2` would be used. Be aware that an **n** value like "2" is *multiplied* by the default number of columns to come up with an effective total number of columns being shifted.
2. Press Enter
3. The line data is shifted the requested number of positions. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift a block of lines toward the left side of your display:

1. Type `[[` in the line command area of the first line to be shifted. Beside the command, type a number other than 1 if you want to shift the block of lines other than 1 indent level. For example, to shift left 2 indent levels `[[2` would be used. Be aware that an **n** value like "2" is *multiplied* by the default number of columns to come up with an effective total number of columns being shifted.
2. Type `[[` in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first `[[` and the second `[[` if necessary.
3. Press Enter
4. The lines that contain the two `[[` commands and all of the lines between them are indent shifted to the left. Characters shifted past the current [BOUNDS](#) setting are deleted.

The [BOUNDS](#) setting limits column shifting. If you shift columns beyond the current [BOUNDS](#) setting, the editor deletes the text beyond the [BOUNDS](#) without displaying a warning message.

]**Indent Shift Right**

Syntax

```
]
]n
]]
]]n
```

Operands

n	A number that tells the editor how many <i>indent increments</i> to shift. If you omit this operand, the default is 1.
----------	--

Description

The **]**Indent Shift Right**** line command moves characters on a line to the right without altering their relative spacing. Characters shifted past the current [BOUNDS](#) setting are deleted.

The command shifts in **indent increments**, where the indent increment is specified by the Options -> General value specified by the "Default # cols for Data shift" entry. This value is treated as the size of each indent level. If an **n** value is specified, the number of **columns** shifted will be **n** times the Options setting. For example, if the Options setting is 3, and a command **]]4** is entered, the lines will be shifted right **12** columns.

For comparison, an Indent Shift respects column boundaries and manages data the same way that a Column Shift does. An Indent Shift may thus be thought of as a specialized type of Column Shift.

To Indent Shift one line toward the right side of your display:

1. Type **]** in the line command area of the line to be shifted. Beside the command, type a number other than 1 if you want to shift the line other than 1 indent level. For example, to shift right 2 indent levels **]]2** would be used. Be aware that an **n** value like "2" is *multiplied* by the default number of columns to come up with an effective total number of columns being shifted.
2. Press Enter
3. The line data is shifted the requested number of positions. Characters shifted past the current [BOUNDS](#) setting are deleted.

To Column Shift a block of lines toward the right side of your display:

1. Type **]]** in the line command area of the first line to be shifted. Beside the command, type a number other than 1 if you want to shift the block of lines other than 1 indent level. For example, to shift right 2 indent levels **]]2** would be used. Be aware that an **n** value like "2" is *multiplied* by the default number of columns to come up with an effective total number of columns being shifted.
2. Type **]]** in the line command area of the last line to be shifted. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **]]** and the second **]]** if necessary.
3. Press Enter
4. The lines that contain the two **]]** commands and all of the lines between them are indent shifted to the right. Characters shifted past the current [BOUNDS](#) setting are deleted.

The [BOUNDS](#) setting limits column shifting. If you shift columns beyond the current [BOUNDS](#) setting, the editor deletes the text beyond the [BOUNDS](#) without displaying a warning message.

Primary Commands

Descriptions of the following Primary Commands are contained in this section:

Note: You can create your own primary-like commands by using the Command Macro facility. You store a series of primary commands in a file and perform them by typing the Macro name. A Command Macro command is entered on the command line just like a built-in primary command. See [Macro Support](#) for more information.

ACTION	Request automatic SAVE/VSAVE be performed
ADD	Add a text line
APPEND	Add text to end of lines
AUTOBKUP	Control backup creation
AUTOCAPS	Control auto-capitalization of language keywords
AUTOSAVE	Control automatic file save defaults
BOTTOM	Scroll to bottom of file
BOUNDS	Set edit boundaries
BROWSE	Open a file for browse (read-only) access, no changes allowed
CANCEL	Cancel edit session without file save
CAPS	Set keyboard CAPS mode
CASE	Control default literal case handling
CHANGE	Change a data string
CLIP	Open a new tab using clipboard data (either Windows or a Private clipboard)
CLONE	Open an un-named edit using an existing file
CMD	Execute another Program or Command
COLLATE	Specify display/sort collating sequence
COLS	Control visibility of top Columns line
COMPRESS	Compress duplicate strings
COPY	Include an external file
CREATE	Create an external file
CRETRIEV	Conditional Retrieve
CUT	Cut data to the clipboard
DELETE	Delete selected lines
DIR	Display folder containing this file in File Manager
DO	Execute the contents of a DO file
DOWN	Scroll downward in the data
DROP	Delete selected lines in a Tag group
EDIT	Open a file for editing
END	End the edit session
ENUMWITH	Change Increment for Enumerate Functions
EOL	Alter end-of-line mode
EXCLUDE	Exclude lines from the display (Edit mode)
EXCLUDE	Exclude files from the display (File Manager mode)
EXIT	Terminate SPFLite session
FAVORITE	Add current file to a Favorite list
FF	Find in Files
FIND	Find a character string

<u>FLIP</u>	Reverse exclusion status of lines
<u>FOLD</u>	Display text in Uppercase only
<u>GLUEWITH</u>	Specify a join string for Glue operations
<u>HELP</u>	Display the Help file
<u>HEX</u>	Set HEX display mode ON or OFF
<u>HIDE</u>	Hide excluded lines
<u>HILITE</u>	Control text highlighting options
<u>JOIN</u>	Join lines using Find / Change strings
<u>KEEP</u>	Delete specific lines in a TAG group
<u>KEYMAP</u>	Display keyboard settings dialog
<u>LC</u>	Lower-case a range of lines
<u>LEFT</u>	Scroll leftward in the data
<u>LOCATE</u>	Scroll the display to a specified line
<u>LRECL</u>	Specify record length
<u>MAKELIST</u>	Create a FILELIST from the current display in File Manager
<u>MARK</u>	Turn MARK lines ON or OFF
<u>MEDIT</u>	Add a file to a Multi-Edit session
<u>NDELETE</u>	Delete lines where string is not found
<u>NFIND</u>	Find lines where string is not found
<u>NFLIP</u>	Reverse exclusion status of lines where string is not found
<u>NEXCLUDE</u>	Exclude lines where string is not found
<u>NREVERT</u>	Revert user line status when string not found
<u>NSHOW</u>	Show (unexclude) lines where string is not found
<u>NULINE</u>	Mark User line status when string not found
<u>OPEN</u>	Edit another file in a new session
<u>OPTIONS</u>	Set editor global options
<u>PAGE</u>	Set Profile PAGE mode ON or OFF
<u>PASTE</u>	Paste data from the clipboard
<u>PREPEND</u>	Add text to the beginning of line(s)
<u>PRESERVE</u>	Control handling of trailing blanks
<u>PRINT</u>	Send selected lines to the printer
<u>PROFILE</u>	Display current file profile values
<u>PTYPE</u>	Enter PowerType mode
<u>QUERY</u>	Display a single Profile setting
<u>RCHANGE</u>	Repeat change
<u>RECALL</u>	Recall (Open) a favorite FILELIST
<u>RECFM</u>	Set record format
<u>REDO</u>	Redo an UNDO action
<u>RELOAD</u>	Reload current edit file from disk
<u>RENAME</u>	Rename the current edit file
<u>REPLACE</u>	Replace a file
<u>RESET</u>	Reset (Edit mode)
<u>RESET</u>	Reset (File Manager Mode)
<u>RETF</u>	Recall commands in a forward direction
<u>RETRIEVE</u>	Recall previous commands
<u>REVERT</u>	Revert User line status
<u>RFIND</u>	Repeat the find command

<u>RIGHT</u>	Scroll rightward in the data
<u>RLOC</u>	Repeat last LOCATE command
<u>RLOCFIND</u>	Repeat most recent FIND or LOCATE command
<u>RUN</u>	Directly execute the current Edit script
<u>SAVE</u>	Save data and continue edit
<u>SAVEALL</u>	Save all current tabs
<u>SAVEAS</u>	Save data as a new file and switch to it
<u>SC</u>	Sentence-case a range of lines
<u>SET</u>	Set a command variable
<u>SETUNDO</u>	Control UNDO levels
<u>SHOW</u>	Show (unexclude) lines where a string is found
<u>SORT</u>	Sort the edit data
<u>SOURCE</u>	Specify character encoding
<u>SPLIT</u>	Split lines using Find / Change strings
<u>START</u>	Set initial file position option.
<u>STATE</u>	Control edit state saving
<u>SUBARG</u>	Set default SUBMIT arguments
<u>SUBCMD</u>	Set alternate command for SUBMIT
<u>SUBMIT</u>	Pass lines to an external command file
<u>SWAP</u>	Switch to Previous or Next Tab
<u>TABS</u>	Turn TABS on or off
<u>TAG</u>	Alter TAG status of a selection of lines
<u>TC</u>	Title-case a range of lines
<u>TOP</u>	Scroll to the top of the file
<u>UC</u>	Upper-case a range of lines
<u>ULINE</u>	Mark lines as User lines
<u>UNDO</u>	Undo changes
<u>UP</u>	Scroll upward in the data
<u>VIEW</u>	View a file in read-only mode.
<u>VSAVE</u>	Perform a virtual save
<u>WDIR</u>	Open Windows Explorer for this file folder.
<u>XSUBMIT</u>	Submit an external file
<u>XTABS</u>	Control handling of incoming tabs

Primary Command Notation Conventions

The syntax of the edit primary commands uses the following notation conventions:

UPPERCASE	Uppercase commands or operands must be spelled as shown (in either uppercase or lowercase).
lowercase	Lowercase operands are variables; substitute your own values
Brackets []	Operands inside brackets [] are optional; the brackets themselves are not typed. When brackets are highlighted as [] they are to be literally typed.
Stacked operands	Stacked operands show two or more operands from which you can select. If you do not choose any, the Editor uses the default operand.
Braces { }	Braces { } show two or more operands from which you must select one.
OR	The OR symbol shows two or more operands from which you must select one.

Line Control Range Specification

Many of the primary commands support line range operands to specify the specific lines which the command is to work on. Rather than repeat these details in each command syntax diagram, whenever the parameter **line-range-operand** appears, the following operands can be substituted.

Syntax

```
[  
  { .start-label | .start-line-number | Relative-start-label }  
  
  [ AND | OR ]  
  
  [ { .end-label | .end-line-number | Relative-end-label } ]  
]  
  
[  
  { :tag | Relative-tag }  
]
```

Alternative Line Command Specification - CC and MM Blocks

In addition to the line number ranges being specified on the command line, all Primary commands which accept line-range-operand format will also accept the line range via line commands. The [C/CC](#) and [W/MM](#) line commands can be used to mark the lines to be used and then the primary command entered without a line range specification.

Single character text selection operands

An optional shortcut method is available for entering the line range operands if suitable. When a text area is selected via a mouse-drag operation or via keyboard Shift-arrow keys, the line and column range of the selected text is remembered for future re-use. If you enter the single character **#** in the command line, it will be substituted with the start/end line numbers of the select block. e.g. **#** could end up as **.20 .30** if that were the previous selected line range.

Operands

start-line-number This should specify the starting line number of a range. This will be either a simple numeric value, or, for those primary commands like **FIND** and **CHANGE** that also support column range operands, the value should be a dotted numeric to distinguish it from the column operand(s). e.g. **.123** would describe line 000123.

start-label This should be a valid label of an existing line to use as the start of range. e.g. **.from**.

relative-start-label This should be a properly formed relative label. A relative label is one which, between the initial period and the first character of the label name, contains one of the conditional operands of **<**, **>**, **<=**, **>=**, ****, **-**, or **<>**. e.g. **.<ABC** or **.<>DD**.

The conditional operands have their normal meanings of equal, less than, greater than, less than or equal, greater than or equal, and not equal.

Note: that when both a start-label and an end-label are specified, and no relative operands are coded, the start-label is assumed to be coded as **>=** and the end-label coded as **<=**. to match prior SPFLite line range conventions. Together with the default of **AND** (next topic), this provides the normal range request of 'all lines from :A thru :B.

AND | OR

You may optionally code **AND** or **OR** between the start and end label references. If you code neither, **AND** is always assumed.

This enables you to code requests such as:

.<ABC OR .>DEF

which would request all lines **before** .ABC and all lines **after** .DEF

end-line-number

This should specify the ending line number of a range. This will be either a simple numeric value, or, for those primary commands like **FIND** and **CHANGE** that also support column range operands, the value should be a dotted numeric to distinguish it from the column operand(s). e.g. **.456** would describe line 000456.

end-label

This should be a valid label of an existing line to use as end of range. e.g. **.to**.

relative-end-label

This should be a properly formed relative label. A relative label is one which, between the initial period and the first character of the label name, contains one of the conditional operands of **<**, **>**, **<=**, **>=**, ****, **-**, or **<>**. e.g. **.<ABC** or **.<>DD**.

The conditional operands have their normal meanings of equal, less than, greater than, less than or equal, greater than or equal, and not equal.

:tag

This should be a valid tagname in the file. The 'range' of lines processed will be ALL lines which are currently tagged with the specified tagname. e.g. **:XXX** would refer to **all** lines which are marked with the **:XXX** tagname.

relative-tag

This should be a properly formed relative tag. A relative label is one which, between the initial colon and the first character of the label name, contains one of the conditional operands of **=**, ****, **-**, or **<>**. e.g. **:=ABC** or **:<>DD**. Note that because tag-names can be present on multiple lines in the file, only equal and not equal conditions can be specified.

The conditional operands have their normal meanings of equal and not equal.

Description

The following examples should make this much clearer. They are shown using a **FIND** command, but line range processing applies to many of the Primary commands:

FIND "ABC" 1 10 .AAA .BBB

Find string "ABC" in columns 1 to 10 on lines from the line labeled .AAA to the line labeled .BBB.

FIND "DEF" 3 30 .110 .120

Find string "DEF" in columns 3 to 30 on lines 110 thru 120.

FIND "GHI" 3 30 .110 .120 :T

Find string "GHI" in columns 3 to 30 on lines tagged with :T, in lines 110 thru 120.

FIND "JKL" .<PROC

Find string "JKL" anywhere on lines from the top of file up to the line preceding the line labeled .PROC. i.e. the range requested is all lines whose line number is **less than** the line labeled .PROC.

FIND "MNO" 3 8 .>=CC .<=DD

Find string "MNO" in columns 3 to 8 on lines from the line labeled .CC to the line labeled .DD inclusive.

FIND "PQR" :XTAG

Find string "PQR" anywhere on any line currently tagged with the tag-name :XTAG.

FIND "STU" :¬BG

Find string "STU" anywhere on any line **not** currently tagged with tagname :BG.

The tag notation :¬BG can also be specified as :<>BG or :\BG , and all of these have the same meaning.

Additional Criteria

Many of the primary commands provide additional selection abilities (like the **X|NX** and **U|NU** keyword operands) which work in conjunction with the normal action of the line range operands described above. Check the detailed description of the individual primary command.

Color Selection Criteria Specification

Many primary commands support selection criteria based on the highlight color of selected text. Rather than repeat these details in each command syntax diagram, whenever the parameter **color-selection-criteria** appears, the following operands can be used.

This means you can choose to find character strings based on whether they are, or are not, displayed in one or more specific colors.

Color operands are used primarily on FIND and CHANGE commands. Just as you have a choice whether you want to just find data or change it, you have a choice as to whether you just want to look for data in some given color, and/or if you want to change its color. Note that finding vs. changing the **color** of your data is performed **independently** of finding or changing the **contents** of your data.

This ability gives you the most flexibility, because the two actions are not dependent on each other. That is, you can change the color of text using a FIND command, or you can use a CHANGE command to alter the data contents of strings that were found to be of some color without also changing that color. It's up to you how you want to manage the color of your data.

Be aware that **space characters** in a file are ordinary data, and can have a color associated with them. For example, if you issue a command like **FIND "ABC DEF" RED**, the string will only be found if the **entire** string is RED, **including the space in the middle**. See [Shifting Data](#) for information on how this can affect Data Shifting commands involving spaces with nonstandard colors.

Note: Color **selection** means that you are **searching** for data that either does, or does not, exist in some particular color. Color selection occurs when you use a "simple" color name like **RED**, or a "negative" color name like **-RED**.

When you use a color name with a + plus sign, like **+RED**, it means you want to **change** the color of any data you are working with. See the next section, [Color Change Request Specification](#), for more information on how to do this.

Note: Color **selection** criteria and a color **change** request can often be combined in the same command, but **not in all cases**. Refer to the syntax of the particular command you are interested in to see which color operands, if any, are supported. Specific color operands are supported only where it makes logical sense to do so.

Standard Color Names

Throughout this description various color names are used. There are 15 fixed color names (**BLUE, GREEN, YELLOW, RED, BLACK, NAVY, TEAL, VIOLET, ORANGE, GRAY, LIME, CYAN, PINK, MAGENTA** and **WHITE**). These are all specified in [Options -HiLites](#). (You are free, if you so choose, to have the standard color name **RED** actually display purple on the screen, or for **BLUE** to display gray, but for most users that would be very confusing).

All names are used in the same way.

As well as the names described above, there are also:

SOLID and **STD**

where **STD** means the standard Foreground / Background colors you have chosen for Text in the Schemes tab of SPFLite Global Options
and **SOLID** means the search string data is entirely of one color, which may be any of 15 available colors.

Syntax

```
[ colorname ] [ STD ] [ SOLID ] ...  
|  
[ -colorname ] [ -STD ] [ -SOLID ] ...
```

Operands

color-names You may specify any one of the defined color names. A color name causes a search for text that consists entirely of the named color. For example, **F "ABC" RED** means find the string "ABC" only if it is entirely RED.

You cannot combine "simple" and "negative" color names on the same command.

-color-names You may specify any one of the defined color names. When color names are entered with a prefix character of - it means to locate text that does not **not** consist entirely of the named color. For example **F "ABC" -RED** would look for ABC only when ABC is not entirely RED.

You cannot combine "simple" and "negative" color names on the same command.

SOLID If **SOLID** is included in the request, it means the found string must be entirely highlighted in one color. For example, **F "ABCD" SOLID** would find the string "ABCD" only if it is found entirely one color, without regard to what particular color that might be. A string **ABCD** highlighted half RED and half BLUE would be ignored because it's not one "solid" color.

SOLID is a request to search for a string entirely in one color, including **STD**.

-SOLID If **-SOLID** is included in the request, it means the found string must **not** be entirely highlighted in one color. For example, **F "ABCD" -SOLID** would find the string "ABCD" only if it is found only partially colored, or colored in more than one color, like **ABCD**.

Description

The following examples should help. They are shown using a **FIND** command, but color selection criteria applies to many of the Primary commands:

FIND "ABC" 1 10 BLUE

Find string "ABC" in columns 1 to 10 only if highlighted in BLUE.
That is, find **ABC**

FIND "DEF" .110 .120 GREEN

Find string "DEF" on lines 110 thru 120 only if highlighted in GREEN.
That is, find **DEF**

FIND "GHI" SOLID

Find string "GHI" anywhere in the file if it is highlighted in any single color or in the default standard color.

FIND "JKL" -BLUE

Find string "JKL" in the file as long as it is not highlighted entirely in BLUE.
That is, if the string is colored as **JKL** it will **not** be found; otherwise it **will** be found.

Additional Criteria

Many of the primary commands provide additional selection abilities (like **X** or **NX**, **U** or **NU** operands) which work in conjunction with the normal action of the line range operands described above. Check the detailed description of the individual primary command.

Color Change Request Specification

Text can be selected manually (either with the mouse or keyboard) and color highlighted using the (Pen/xxxx) keyboard primitives. (Pen/Blue), (Pen/Green), etc. This is fine for ad-hoc marking of text, but when you'd like to mark, say, all occurrences of a string this would be a large task.

Many of the primary commands which support selection criteria now also support the automatic highlighting of the text found during the operation. This is performed **in addition to** the basic purpose of the command. The operands described here allow you to request this highlighting. Rather than repeat these details in each command syntax diagram, whenever the parameter **color-change-request** appears, the following operands can be substituted.

Color operands are used primarily on FIND and CHANGE commands. Just as you have a choice whether you want to just find data or change it, you have a choice as to whether you just want to look for data in some given color, and/or if you want to change its color. Note that finding vs. changing the **color** of your data is performed **independently** of changing the **contents** of your data.

This ability gives you the most flexibility, because the two actions are not dependent on each other. That is, you can change the color of text using a FIND command, or you can use a CHANGE command to alter the data contents of strings that were found to be of some color without also changing that color. Its up to you how you want to manage the color of your data.

Note: Color **selection** criteria and a color **change** request can often be combined in the same command, but **not in all cases**. Refer to the syntax of the particular command you are interested in to see which color operands, if any, are supported. Specific color operands are supported only where it makes logical sense to do so.

Standard Color Names

Throughout this description various color names are used. There are 15 fixed color names (**BLUE, GREEN, YELLOW, RED, BLACK, NAVY, TEAL, VIOLET, ORANGE, GRAY, LIME, CYAN, PINK, MAGENTA and WHITE**). These are all specified in [Options -HiLites](#). (You are free, if you so choose, to have the standard color name **RED** actually display purple on the screen, or for **BLUE** to display gray, but for most users that would be very confusing).

All names are used in the same way.

As well as the names described above, there is also:

+STD

where **+STD** means the standard default Foreground / Background colors you have chosen for normal text.

Note: The + plus sign should be read as "change", **not** as "positive". The operand **+RED** is **not** the opposite of a "negative" search color like **-RED**. **-RED** means to **search** for data which is not RED, while **+RED** means to **change** the color of data to RED.

Syntax

```
[ +colorname ] [ +STD ]
```

Operands

+color-name	You may specify at most only one color name . For example F "ABC" +RED ALL means find all the strings "ABC" and add RED highlighting to the string. It would not make sense to use multiple change-color names, since a given text string can only be in one color at any given time.
--------------------	---

Description

The following examples should help.

FIND "ABC" 1 10 +BLUE

Find string "ABC" in columns 1 to 10 and highlight it in **BLUE**.

Here, a "change" occurs to the color of the data, even though there is no change to the contents of the data. For that reason, we use a **FIND** rather than a **CHANGE** command.

FIND "DEF" .110 .120 GREEN +YELLOW

This example uses both a color-selection-criteria and a color-change-request criteria.

Find the string "DEF" on lines 110 through 120 only if it currently is entirely **GREEN**, and if found, change its color to **YELLOW**.

CHANGE "GHI" "XYZ" +BLUE ALL

Change all strings "GHI" anywhere in the file to "XYZ" and set the changed string to **BLUE**.

Note that because no "simple" name like **RED** and no "negative" name like **-RED** is specified, the original color of any **GHI** strings is ignored. The command will find all **GHI** strings without regard to their original color.

FIND "DEF" .110 .120 -STD +STD

This example uses both a color-selection-criteria and a color-change-request criteria.

Find the string "DEF" on lines 110 through 120 only if it currently is **not** the standard default color, and if found, change its color back so that it **is** the default color.

Note that if the **-STD** operand above was omitted, the **DEF** data would still end up set to the standard color the same way, but some text that was **already** in the standard color would get needless "changed" to the color it already was. For small files, this may not matter, but for very large files you will likely want to restrict the color changing operation to only the data that actually needs to be changed.

Additional Criteria

Many of the primary commands provide additional selection abilities (like **X** or **NX**, **U** or **NU** operands) which work in conjunction with the normal action of the line range operands described above. Check the detailed description of the individual primary command.

Clearing colors

If you need to clear the colors from a file, there are various ways to do this.

To clear colors from the entire file, issue the primary command **RESET COLOR**. See [RESET - Reset \(Edit Mode\)](#) for more information.

Note: A plain **RESET** without **COLOR** doesn't clear colors, because if you went through the trouble of coloring your text, and you issued a plain **RESET** (say, to clear out some excluded lines), you'd be really annoyed if all your colors went away. That's why SPFLite generally doesn't have its plain **RESET** command do more things than IBM's ISPF does. People would not be expecting it, and it would cause too many problems. Plus, colors are not reset very often, whereas plain **RESET** commands get issued all the time.

Suppose you wanted to reset a block of text you highlighted with the mouse. (Be careful here not to confuse "highlighting" with "coloring". When we say highlighting, we mean "text selection" using the mouse or with the shift-arrow keys. Highlighting just means "this is a piece of text I want to reference". It has nothing to do with colors.)

Now, if you highlight some text, you can alter the color of it using keyboard primitive functions. (Yes, in the Screen Options menu, it says "Highlight Red" etc. and the Help talks about Virtual Highlighting Pens. Unfortunately, there are not enough English words to say everything we'd like to say as uniquely as we'd like to say it.)

For example, you could define the following [KEYMAP](#) entries with virtual highlighting pen functions:

Ctrl-Shift R = (Pen/Red)

Ctrl-Shift B = (Pen/Blue)

Ctrl-Shift G = (Pen/Green)

Ctrl-Shift Y = (Pen/Yellow)

Ctrl-Shift S = (Pen/Std)

Let's say you have some string that is red. You can highlight it with the mouse, and assuming your keys are set up as shown, you could press Ctrl-Shift S, and the text color will be changed back to the "standard" non-colored one.

ACTION - Request periodic SAVE/VSAVE

Syntax

ACTION	['n'] [SAVE VSAVE]
---------------	--

Operands

'n' The desired interval between automatic SAVE/VSAVE. (a '0' zero value indicates **no** automatic saves are desired.

SAVE | **VSAVE** Which type of save you desire, a full SAVE or a temporary VSAVE.

Description

If **0** is specified, it is treated as a request for **no** automatic saving.

A numeric value entered it is treated as the number of edit interactions to occur between automatic saves. (See Edit Interactions below)

The value is stored as part of the PROFILE options which are maintained individually by file type.

Processing

Depending on your work style, it can be beneficial to have a periodic SAVE or VSAVE command issued automatically. During long edit sessions it is easy to forget to save periodically. When ACTION is activated, SPFLite will automatically perform a SAVE or VSAVE function at your specified interval. Note this interval is **not** an elapsed time measurement, the interval is not 'minutes' or other time interval. Please read the following description carefully.

Edit Interaction

To specify the operand value for ACTION, an understanding of what an Edit Interval **is** is important.

An *Edit Interval* is counted when the file being edited is changed in any way and the Enter or a Function key is pressed. Simply pressing Enter, or a function key which does not alter the text in any way (like PgDn/PgUp) will not count as an interaction. Similarly, making multiple changes to the text, issuing line commands and/or primary commands like CHANGE, all at one time still only counts as **one** interaction.

ADD - Add a text line

Syntax

ADD	string
	line-reference

Operands

string The string option is required, and can only be a simple string. (Quoted or unquoted) i.e. No [Picture strings](#) or [Regular expression strings](#). This string will become the complete contents of the inserted text line. Any of the valid quote types may be used for quoted strings.

line-reference Specified the line **after which** the specified string of text is to be inserted as a line. This should be a label reference, either a label name like `.ABC` or a pseudo-label line number like `.123`.

Note that a macro-oriented line-pointer reference beginning with an `!` is also supported.

Description

Although this command can be issued from the normal command line, it is primarily intended to be a convenient tool for macros to use when inserting text lines. In a programmable macro, it would be issued using an **SPF_Cmd** function. You can also map a key to ADD a line.

ADD will insert the specified text string as a new line immediately following the provided line-reference.

Example use of the ADD command

To insert a line containing the string `/*-----*/` following line 10.

```
ADD /*-----*/ .10
```

Suppose you wanted to insert a line containing `***` wherever the cursor happened to be, whenever you pressed the F9 key. You would go into KEYMAP and enter the following into the "Normal" entry for F9:

```
ADD '***' .ZCSR
```

APPEND - Add text to end of line

Syntax

```
APPEND           string
                  [ start-column [ end-column ] ]
                  [ line-control-range ]
                  [ color-selection-criteria ]
                  [ X | NX ]
                  [ U | NU ]
                  [ ALL ]
                  [ TOP ]
```

Operands

string	The string option is required, and can only be a simple string. i.e. No Picture strings or Regular expression strings . This string will be appended to the end of every line specified by the other APPEND operands.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be eligible to be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Description

APPEND adds the specified string to all lines specified by the line-control-range, X/NX and ALL operands.

APPEND will work on lines within the line-control-range that are zero-length. This is one technique for converting such lines into lines that are not zero-length.

Example uses of the APPEND command

To append the string ")." to all excluded lines.

```
APPEND ') . ' ALL X
```

To append '---' to lines 10 thru 20 of the file.

```
APPEND '---' .10 .20
```

To append '[135]' to all non-excluded lines in the line range 3 thru 200.

```
APPEND '[135]' nx .3 .200
```

Comparison to Power Typing

In some cases it is possible to append strings to the right-hand side of a column of values, even if the values are of differing lengths, using Power Typing. See [Working with Power Typing Mode](#) for an example of how to do this.

AUTOBKUP - Control Creation of Backups

Syntax

AUTOBKUP	[<u>ON</u> <u>OFF</u>]
-----------------	-----------------------------------

Operands

ON | OFF The desired AUTO-BACKUP status

Description

If **ON** is specified, SPFLite will create a backup copy of any Edited file by storing it with an additional level of '.BKP'. The ".BKP" level will be inserted before the last level. e.g. A Backup of MYDATA.TXT would be saved as MYDATA.BKP.TXT. If the .BKP file already exists, it will be overlaid.

If **OFF** is specified, then no Backup will be created.

The value is stored as part of the PROFILE options which are maintained individually by file type.

AUTOCAPS - Control AutoCaps Mode

Syntax

AUTOCAPS	[<u>ON</u> <u>OFF</u>]
-----------------	-----------------------------------

Operands

ON | OFF The desired AUTOCAPS status

Description

If **ON** is specified, SPFLite will uppercase all language based keywords as specified by the .AUTO colorization file for this Profile. This feature is dependent on the .AUTO support, since the .AUTO file contains the definition of the language keywords. See [Auto Capitalization Support](#) in the Appendix for more details.

If a valid .AUTO file does not exist, or if HILITE AUTO ON is not also set, then setting AUTOCAPS ON will have no effect.

If **OFF** is specified, then AUTOCAPS processing is not performed.

The value is stored as part of the PROFILE options which are maintained individually by file type.

Processing

When active, AUTOCAPS will cause all identified keywords in the associated .AUTO file to be upper-cased in the file display. The keywords are also uppercased as typed.

e.g. Assume both FOR and FOREVER are valid keywords in the current .AUTO file. Then assuming the word 'forever' was typed (in lower-case), it would appear successively as:

```
f
fo
FOR
fore
forev
foreve
FOREVER
```

As this shows, the uppercasing is **not** done by actually changing the text data as keywords are detected, if that were the case the sequence would have looked like:

```
f
fo
FOR
FORe
FORev
FOReve
FOREVER
```

and if you actually wanted to stop with the word 'fore' you would not be too happy to have it as 'FORe' .

So the uppercasing is performed only on what is displayed on the screen. But fear not, when the file is saved, the uppercasing will be 'finalized' and written to the output file. So what you **see** is what you **get**.

AUTONUM - Number lines automatically

Syntax

AUTONUM	[ON OFF]
----------------	---------------------

Operands

ON OFF	The desired AUTONUM status
-----------------	----------------------------

Description

If **ON** is specified, automatic sequence numbering is enabled so that at SAVE or END time, your file is renumbered in the same way as if a RENUM command had been issued.

If **OFF** is specified, then automatic sequence numbering at SAVE or END time is disabled.

The value is stored as part of the PROFILE options.

If neither ON nor OFF is specified, AUTONUM will report the current automatic sequence numbering option in effect.

The automatic sequence numbering operation will renumber the entire file.

Implementation Notes

Automatic AUTONUM sequence numbering will be performed only under the following conditions:

- AUTONUM is ON
- NUMTYPE must be set to a valid numbering type other than NONE
- The file's modification status causes a SAVE to be performed.

AUTOSAVE - Control SAVE Action at END

Syntax

AUTOSAVE	[<u>ON</u> <u>OFF</u>]
	[<u>PROMPT</u> <u>NOPROMPT</u>]

Operands

ON | OFF The desired AUTOSAVE status

PROMPT | NOPROMPT The desired Prompt status

Description

If **ON** is specified (or defaulted), SPFLite will create automatically save the file data when an END command is issued, or when the entire SPFLite window is shut down. See Prompt options below for how Prompt interacts with this setting.

If **OFF** is specified, then SPFLite will not automatically save the file data. See Prompt options below for how Prompt interacts with this setting.

If **PROMPT** is specified, then regardless of the **ON / OFF** status, you will be prompted with an option box where you may manually choose to save or not save the file.

If **NOPROMPT** is specified, then the specified **ON / OFF** action will be performed without any prompt intervention.

The value is stored as part of the PROFILE options which are maintained individually by file type.

AUTOSAVE OFF NOPROMPT Warning

If you set AUTOSAVE to OFF NOPROMPT, an END command (usually assigned to F3), will close your file **without saving any unsaved changes, and will not warn you or prompt you of this fact.**

To avoid data loss, the status line will contain the message, **AUTOSAVE OFF** as a **reminder** that this situation exists. You are not required to change the AUTOSAVE setting when you see this reminder. It simply advises you of this fact, so you do not inadvertently lose data.

BOTTOM - Scroll to Bottom of File

Syntax

```
BOTTOM
```

Operands

None

Abbreviations and Aliases

BOTTOM can also be spelled as **BOT**

Description

The **BOTTOM** command will scroll the edit window to the bottom of the data file. **BOTTOM** is an alias for **DOWN MAX**.

BOUNDS - Set Edit Boundaries

Syntax

BOUNDS	[[left-col] { right-col MAX }]
--------	---

Operands

left-col If the left column boundary is omitted, it is assumed to be 1.

right-col | The right column boundary to be set or the Keyword 'MAX' or + to indicate the maximum record length.
MAX |
+
You cannot specify the same column for both boundaries..

If you specify BOUNDS MAX or BOUNDS 1 MAX, you will see a bounds display on the status line of **Bnds: MAX**. When this is in effect, you can describe this as "edit columns are unbounded" or "the editor is in unbounded mode".

If you specify BOUNDS **n** MAX, where **n** is any value higher than 1, you will see a bounds display on the status line of **Bnds: n to MAX**

Abbreviations and Aliases

BOUNDS can also be spelled as **BOUND**, **BNDS**, **BND** or **BOU**

Description

The BOUNDS primary command provides an alternative to setting the boundaries with the BNDS line command; the effect on the data is the same. However, if you use both the BOUNDS primary command and the BNDS line command in the same interaction, the primary command overrides the line command.

The left column boundary number is optional, and if omitted, it is assumed to be 1.

To reset the boundaries to the default column (1 and Max Record Length):

1. On the command line, type: BOUNDS or BOUNDS MAX
2. Press Enter
3. The boundaries are reset to the defaults.

Note: When the **BOUNDS** setting is anything other than **MAX**, the status line display will show the **BOUNDS** setting in white letters on a red background, like **Bnds: 1 to 40** so that it can't be ignored. This will help users to avoid the unexpected and nonstandard handling of data that occurs when non-default bounds are in effect, if that was not their intent.

A word about the use of BOUNDS

The BOUNDS feature of ISPF is one that IBM did not extensively document, and in practice, mainframe ISPF users do not tend to use this feature very often. Every effort was made to implement BOUNDS in an ISPF-compliant manner. However, it may produce surprising and unexpected results if you are not familiar with

the actions taken by various commands when operating under restricted column BOUNDS. The "surprising and unexpected" aspect is even more of a factor for SPFLite users without a prior mainframe ISPF background.

For many users, you will likely get the most benefit from SPFLite by operating in unbounded mode most or all of the time, and not worry about using BOUNDS unless you have very particular editing requirements.

BROWSE - Open a File for Read-Only

Syntax

```
BROWSE      [ file-name | * ]
            [ .Profile-name ]
```

Operands

file-name | * The name of the file to be browsed.

If a single * is entered, it requests the filename to be fetched from the current clipboard contents.

.**Profile-name** This optional operand allows you to specify an overriding Profile to be used for the Browse session. It simply consists of the name of the desired Profile, preceded by a . (period)

Abbreviations and Aliases

BROWSE can also be spelled as **B** or **BRO**

Description

The **BROWSE** command loads a file into the work space for Browsing. If no file-name operand is specified, a standard Windows File Open Dialog will be presented for you to select a file for Browsing.

BROWSE will not allow you to make **any** modifications to the file. Primary commands, Line Commands and normal typing which would modify the file's data are suppressed. To remind you, the word **Browse** in the left-hand Status Bar box will be displayed as **Browse**. These restrictions prevent you from modifying files that might be important to you, when you wanted to be sure there was no possibility you could change them by accident.

- This represents a change from prior versions. If you need the old behavior of **BROWSE**, in which files could be temporarily changed and saved under a different name, use the [VIEW](#) command instead.
- The use of **BROWSE** and [VIEW](#) now conforms more closely to ISPF usage. However, unlike the "non resident" behavior of **BROWSE** on the mainframe, the SPFLite **BROWSE** will still load your file into memory, and so the same memory considerations that existed in prior SPFLite versions are still present.

If the current working Tab contains data, a new Tab will be opened to hold the Browse session; otherwise the current Tab will be used.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the **BROWSE** command, the default directory used for searching for the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the Path for THAT active file is used as the default for the command.
- If there is NO active file [when the tab header displays (Empty) then the current displayed directory of File Manager will be used.

Overriding Profile

When you wish to use a different Profile from the one indicated by the file's extension, simply provide the alternate profile name, preceded by a period, on the command line. For example, to Browse MYSOURCE.BAS using the TXT profile the command would be

```
BROWSE MYSOURCE.BAS .TXT
```

The use of the alternate profile is for the duration of this session only. It does not alter any permanent Profile processing.

CANCEL - Cancel Edit Session

Syntax

CANCEL	[DELETE PURGE]
---------------	---------------------------

Operands

DELETE | **DEL**

When you issue **CANCEL DELETE**, the edit session is canceled and then the edit file is deleted:

- If the **Delete to Recycle Bin** global option is **enabled**, deletion means that the edit file is moved to the Windows Recycle Bin.
- If the **Delete to Recycle Bin** global option is **disabled**, deletion means that the edit file is **purged**, which is physical deletion without recourse.

PURGE

When you issue **CANCEL PURGE**, the edit session is canceled and then the edit file is purged, which is physical deletion without recourse. The edit file will not be moved to the Windows Recycle Bin, even if the **Delete to Recycle Bin** global option is enabled.

Abbreviations and Aliases

CANCEL can also be spelled as **CAN**

DELETE can also be spelled as **DEL**

Description

CANCEL is especially useful if you have changed the wrong data, or if the changes themselves are incorrect, or were only intended to be temporary. To cancel changes to a file:

On the command line, type:

CANCEL

Press Enter. All unsaved changes in the current edit session are discarded, and the current edit tab is closed.

Notes:

- **CANCEL** clears the Undo stack, so **you cannot UNDO a CANCEL command**. You cannot recover the changes you made if you say **CANCEL**, so be sure this is what you intended.
- If you issue [SAVE](#) and later issue [CANCEL](#), the changes you made before issuing [SAVE](#) are not canceled.

When you issue a **CANCEL DELETE** or **CANCEL PURGE**, you will be presented with a confirmation popup message, asking you to confirm your deletion action. This popup message appears when the [Confirm file deletes](#) checkbox in the [File Manager tab](#) of Global Options is enabled.

For safety reasons, we recommend you enable this option.

You will either see **Recycle** or **Purge** in the popup message, to identify the action.

If you see **Recycle** but you really wanted to **Purge** or vice-versa, cancel the popup message, and either use a different keyword on the **CANCEL** command, or change the [Delete to Recycle Bin](#) checkbox in the [General tab](#) of Global Options so that deleted files are handled in the way you prefer.

CAPS - Set Keyboard CAPS Mode

Syntax

CAPS	[<u>ON</u> OFF AUTO]
-------------	-----------------------------------

Operands

ON | OFF | AUTO The desired CAPS status

Description

The SPFLite **CAPS** setting is independent of the normal Windows keyboard Shift and Caps Lock handling. When **ON** is specified, it requests SPFLite to ensure that any modified line is Uppercased regardless of the current keyboard caps status. For example, **CAPS ON** would be a suitable default for the file types used to edit mainframe JCL job streams which must always be uppercase only.

If **OFF** is specified, then no automatic case conversion will be done.

SPFLite handles CAPS mode differently than IBM ISPF

In ISPF, no matter what you set your **CAPS** mode to, the ISPF editor will examine your file when you open it. If it finds data which is contrary to your **CAPS** setting, it will forcibly change the **CAPS** mode **ON** or **OFF** to conform to whatever is in your data, and that **CAPS** setting will be permanent (until you change it, or ISPF changes it again in the same way). Experience has shown that the way that ISPF handles this can be problematic.

In SPFLite, when you set **CAPS** mode to **ON** or **OFF** for a given file type, it stays that way **until you alone change it**. SPFLite **never** changes the **CAPS** mode from **ON** to **OFF** or vice versa, by itself.

Note that when **CAPS ON** is in effect, while you are typing in new characters on a data line, they will appear in the "Text - High-Intensity" font color. This is done as a visual reminder that the alphabetic data you are entering is being forced to upper case (this is regardless of the Caps Lock key). If that is not what you wanted, issue the **CAPS OFF** command. Using **CAPS ON** is common for mainframe data, while **CAPS OFF** is more typical of PC data.

If you set **CAPS** mode to **AUTO**, the caps mode is "conformant". The file is examined, comparable to how ISPF does it, but in SPFLite this setting is only temporary. When you open the file again, the file examination is repeated. The "on" or "off" displayed in *lower case* is a reminder that the current automatic/conformant caps mode is **temporary**, and is **not** stored in the PROFILE, whereas **CAPS ON** and **CAPS OFF** is stored in the PROFILE.

The **AUTO** part of **AUTO: on** or **AUTO: off** is stored in the PROFILE, but the **on/off** setting itself is **not** stored.

The CAPS AUTO status display depends on the prior CAPS state

The CAPS status line display will change in response to a **CAPS AUTO** command as follows:

- If the current caps mode is already **CAPS AUTO**, issuing it again will cause the CAPS state will toggle between **CAPS AUTO:on** and **CAPS AUTO:off**.
- If the current caps mode is **CAPS ON**, a **CAPS AUTO** command will cause the status line to display

CAPS AUTO:on.

- If the current caps mode is **CAPS OFF**, a **CAPS AUTO** command will cause the status line to display **CAPS AUTO:off**.

The Effect of CAPS AUTO on the Profile

The **AUTO** part of **AUTO: on** or **AUTO: off** is stored in the **PROFILE**, but the **on/off** setting itself is not stored, because it is determined each time the file is opened.

CASE - Set Default String Case Handling

Syntax

CASE	[C T]
-------------	------------------

Operands

C T	The desired CASE status
--------------	-------------------------

Description

A literal used in an SPFLite command line can be specified with an explicit case-sensitivity type code. A literal of **C 'aBc'** requests 'Character-mode' case-sensitive handling, while a literal of **T 'aBc'** specifically requests 'Text-mode' case-insensitive handling.

But how should unquoted strings like **Abc**, and quoted strings without type codes like '**aBc**' be handled?

The **CASE** command allows you to specify how the **search** strings in FIND, CHANGE and similar commands are to be handled when a type of **C** or **T** is not used.

When **C** is specified, all general form literals, even unquoted strings, are handled as if they were specified as **C'string'**.

When **T** is specified, all general form literals, even unquoted strings, are handled as if they were specified as **T'string'**.

The current **CASE** setting is used to determine the case-sensitivity of alphabetic characters used in search Picture strings. The **CASE** setting also controls the default sort order in SORT.

The current setting of **CASE** can be seen by the CASE-code in the middle of the status line.

- If **C** or **C W** appears, **CASE C** is in effect.
- If **T** or **T W** appears, **CASE T** is in effect.

Note that the **CASE** command, and the current **CASE** setting, has no effect in how the **change** string of **CHANGE** commands is processed.

The **CASE** value is stored as part of the PROFILE options which are maintained individually by file type.

CHANGE - Change a Data String

Syntax

```
CHANGE      from-string  to-string
           [ start-column [ end-column ] ]
           [ FIRST | LAST | NEXT | PREV | ALL ]
           [ PREFIX | SUFFIX | WORD | CHAR ]
           [ LEFT | RIGHT ]
           [ TRUNC ]
           [ line-control-range ]
           [ color-selection-criteria]
           [ color-change-request]
           [ DS | CS ]
           [ X | NX ]
           [ U | NU ]
           [ MX | DX ]
           [ TOP ]
```

Operands

from-string	The search string you want to change
to-string	The string you want to replace from-string
start-column	Left column of a range (with end-column) within which the from-string value must be found. If no end-column operand, then the from-string operand must be found starting in start-column.
end-column	Right column of a range (with start-column) within which the from-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of from-string.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of from-string.
<u>NEXT</u>	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of from-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of from-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of from-string.
LEFT	LEFT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the left-most occurrence of search-string is changed, and any other instances on that same line are unchanged.

RIGHT	RIGHT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the right-most occurrence of search-string is changed, and any other instances on that same line are unchanged.
TRUNC	TRUNC will cause the remainder of the line, following the replacement CHANGE string, to be deleted (truncated).
PREFIX	Locates from-string at the beginning of a word.
WORD	Locates from-string when it is delimited on both sides by blanks or other non-Word characters.
<u>CHAR</u>	Locates from-string regardless of what precedes or follows it.
SUFFIX	Locates from-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the from-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
color-change-request	A request can also be made to highlight the string following completion of the command. The full syntax and allowable operands which make up a color-change-request are discussed in "Color Change Request Specification" . Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
CS DS	Specifies the Column Shift / Data Shift mode for this particular CHANGE command. See Effect of CHANGE Command on Column-Dependent Data for the significance of this operand. If not specified, the default CS/DS value for this file's Profile will be used. If a CS or DS is the only operand of the CHANGE command, the specified value will be set as the default for the file's Profile.
MX	MX requests that all lines which DO contain from-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain from-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the second screen

line (as ISPF does) if the line is not on the current screen. If **TOP** is coded, then the line is always repositioned as the **top** line of the screen, regardless of its current location.

Abbreviations and Aliases

CHANGE can also be spelled as **C**, **CHG** or **CHA**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the **CHANGE** command to change one string to another either singly, or for all occurrences. As well optional operands allow you to restrict the column boundaries within which it operates, along with other characteristics of the string searched for.

To change the next occurrence of ME to YOU without specifying any other qualifications:

On the command line type:

```
CHANGE ME YOU
```

Press Enter. This command changes only the next occurrence of the letters ME to YOU.

Since no other qualifications were specified, the letters ME can be:

- Uppercase or a mixture of uppercase and lowercase. (Depends on **CASE** setting, see the [CASE](#) command)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- In an excluded line or a nonexcluded line
- Anywhere within the current boundaries.

To change the next occurrence of ME to YOU, but only if the letters are uppercase:

On the command line type:

```
CHANGE C 'ME' YOU
```

Press Enter. This type of change is called a case sensitive string change (note the C that precedes the search string) because it changes the next occurrence of the letters ME to YOU only if the letters are found in uppercase. However, since no other qualifications were specified, the change occurs no matter where the letters are found, as outlined in the preceding list.

For more information, including other types of search strings, see [Finding and Changing Data](#) and [Specifying a Picture or Format String](#).

The following example changes the first plus in the file to a minus. However, the plus must be the first character of a word:

```
CHANGE '+' '-' FIRST PREFIX
```

The following example changes the last plus in the file to a minus. However, the plus must be the last character of a word; and it must be found on an excluded line:

```
CHANGE '+' '-' LAST SUFFIX X
```

The following example changes the first plus that precedes the cursor position to a minus. However, the character must be a stand alone character (not part of any other word); it must be on a nonexcluded line; and it must exist within columns 1 and 5:

```
CHANGE '+' '-' PREV WORD NX 1 5
```

The following example changes all words ABC to DEF from the line at label .A to the line at label .B. If the word ABC appears more than once in any given line, only the left-most one is changed, and any additional occurrences of ABC on the same line are ignored:

```
CHANGE LEFT ABC DEF .A .B ALL
```

The following example changes all strings "XYZ" which are highlighted in BLUE to "PQR". The resulting string will remain colored BLUE.

```
CHANGE ALL "XYZ" BLUE "PQR"
```

The TRUNC keyword will cause the CHANGE command to truncate the line after point where the change-string replaces the find-string. The following example changes the characters "END)" " to "END ." and deletes all remaining characters on the line.

```
CHANGE "END)" "END ." TRUNC
```

which will alter the line

```
COMING TO AN END) BUT NOT BEFORE
```

into

```
COMING TO AN END.
```

When using TRUNC, the change string can be a zero-length string, which results in truncating the find-string and everything that follows it. The following example deletes the character ")" " and all remaining characters on the line.

```
CHANGE ")" "" TRUNC
```

which will alter the line

```
COMING TO AN END) BUT NOT BEFORE
```

into

```
COMING TO AN END
```

Note: When the **CHANGE** search operand is a Regular Expression (a string with an R type code) and reverse-order searching is done with **PREV** or **LAST**, only the left-most occurrence on any given line is changed. That is, the command

```
CHANGE R'ABC' 'XYZ' PREV
```

is treated as if it were specified as

```
CHANGE LEFT R'ABC' 'XYZ' PREV
```

and

```
CHANGE R'ABC' 'XYZ' LAST
```

is treated as if it were specified as

```
CHANGE LEFT R'ABC' 'XYZ' LAST
```

This limitation stems from the regular-expression engine used by SPFLite.

CLIP - Open New File Tab with Clipboard Data

Syntax

CLIP	[clipboard-name]
-------------	---------------------------

Operands

clipboard-name The optional *clipboard-name* operand allows you to open a [Private Named Clipboard](#) rather than the standard Windows clipboard. If not present, the standard Windows clipboard is assumed.

Description

The **CLIP** command will open a new edit Tab and load into it the current contents of the specified Clipboard.

When the Clip window is closed with the **END** command, the contents at that point will be returned to the specified Clipboard.

If the operand is omitted, the **CLIP** command will assume the standard Windows clipboard is to be used.

When **CLIP** is used without the clipboard-name operand, its action is identical to **-Clip** mode when invoked from the startup command line

Note: It is important to understand that a CLIP Edit session **copies** the clipboard into an SPFLite-controlled edit session dedicated to clipboard editing, and then **copies it back** to the actual clipboard when you're done.

Remember: The clipboard, and the clipboard edit session, are **not** the same thing.

While you are **in** the CLIP Edit session, you are not actually modifying the Windows clipboard - only a copy of what it was, prior to the CLIP Edit session beginning. What that means is that, while you are in a CLIP Edit session, you could jump outside of it, to another SPFLite edit tab, or outside of SPFLite itself, then copy **more** data to the Windows clipboard, and paste **that** into your CLIP Edit session, and you could continue doing that process as many times as you like. Only once you're done will the edit session be copied back to the Windows clipboard. This fact can make for some very interesting and powerful editing techniques.

See [Windows Clipboard, Cut and Paste](#) for more information.

CLONE - Open an Unnamed Edit Using an Existing File

Syntax

CLONE	[filename *] [.prof-name]
--------------	--

Operands

filename *	The name of the file to be cloned. If the filename operand is omitted, and the command is issued from the Edit tab for a file, the filename being edited will be cloned from the current disk version, NOT the edit session contents. If a single * is entered, it requests the filename to be fetched from the current clipboard contents.
.prof-name	If entered, you may specify a different Profile name to use. If omitted, the Profile will be assigned in the normal manner, based on the file extension.

Description

The CLONE command will open a new edit session using a copy of the named file. You will be prompted for a new filename to be used for this cloned file.

When you use CLONE, the cloning action uses the **current on-disk** version of the filename, **not** the contents of the file that might exist in any edit session. If you want the cloned session to include the current state of the edit session and there are unsaved changes, then you must [SAVE](#) the current session before using the CLONE command.

Because CLONE copies an existing on-disk copy of a file, an on-disk copy of the file must exist.

That means you cannot CLONE the following:

- a NEW Empty File
- a SET Edit session
- a CLIP Edit session

Cloning a file from its contents on disk is the same action that takes place when the Clone line command C is used in File Manager.

Cloning the current Edit / Browse file

If you wish to CLONE the current Edit / Browse session, simply enter CLONE with no operands. .

CMD - Execute Another Program or Command

Syntax

CMD	command-line
-----	---------------------

Operands

command-line The command line to invoke the new program as it would be entered to the system CMD processor.

Description

The **CMD** command will issue the command to start the named program or command. It does not wait for completion of the program nor check for successful completion status. Because **CMD** calls the Windows **CMD.EXE** program, any Windows command-line command, like **DIR**, can be directly issued, like **CMD DIR**. If the command-line has options, you can type them directly as you would on a Windows command line without requiring quotes, like **CMD DIR C:\MYPATH\ABC.***.

The **CMD** command uses the same PATH environment variable value as Windows does in determining where the command being run is located. If your program or script is in a directory named by the PATH, you do not need a fully-qualified command name. Otherwise, it must be properly qualified in order to be found.

If you are unsure of the contents of the PATH environment variable, you can issue the command **CMD SET PATH** from SPFLite, and it will display the PATH in effect for the CMD command. Press Enter when you are finished looking at the output to return to SPFLite.

When the **CMD** command is issued, SPFLite must determine the location of the current working directory - the directory used to locate data files (not executables or scripts) that are not specified as fully-qualified file names. It does this as follows:

- If **CMD** is issued from an Edit or Browse session with a named file, the file path of that file becomes the current working directory.
- If **CMD** is issued from an Edit of a Cloned file, the file path of the original file becomes the current working directory.

Any path-dependent or unqualified file names used in the command should take this into account.

Command Window Closing

You may control whether the command window opened by the **CMD** command remains open at the completion of the command. There is an option under [Options => Submit](#) which allows you to specify your choice. In the CMD parameters box on the Submit tab enter **/C** to close the window on completion, or **/K** to keep the window open.

COLLATE - Specify Display Character set

Syntax

COLLATE	[ANSI <i>source-name</i>]
----------------	--------------------------------------

Operands

ANSI Requests use of the standard ANSI character set

source-name Requests use of the codepage called *source-name*.

Abbreviations and Aliases

ANSI can also be spelled as **ASCII**

Description

There are times when it can be useful to “pretend” that the edit data is another codepage than that specified by the **SOURCE** setting. For example, if you were looking at a SYSOUT file from a mainframe (let's say, an EBCDIC-based MVS job stream file containing JCL, compiler listings and user program output), which had been translated back to ANSI when sent to the PC, it is difficult to treat this as EBCDIC data, since it no longer is encoded as EBCDIC.

By specifying **COLLATE EBCDIC**, when display mode is set to **HEX**, you will see EBCDIC hex equivalents rather than ANSI. Similarly, the **SORT** command will use EBCDIC for the collating sequence rather than ANSI.

The **COLLATE EBCDIC** command allows you to have the “experience” of treating a file as though it were EBCDIC, even when it really isn't.

The source operand can be any value for which a valid ***source-name.SOURCE*** file exists in the SPFLite data directory. SPFLite is distributed with a default translation table called **EBCDIC.SOURCE**. You can create additional translation tables to meet specific requirements, including tables which translate from one variant of ASCII to another.

It is possible, though not recommended, to directly modify the standard **EBCDIC.SOURCE** file.

The **COLLATE** setting by default is the same as the **SOURCE** setting. It is associated with individual file types and is stored in the Profile for that file type.

COLS - Control Visibility of Top Columns Line

Syntax

COLS	[<u>ON</u> OFF]
------	---------------------

Operands

ON | OFF The desired COLS status

Abbreviations and Aliases

COLS can also be spelled as **COL**

Description

If **ON** is specified, SPFLite will display a permanent COLS line at the top of the screen. This line will remain in place regardless of the file position being viewed.

If **OFF** is specified, then no COLS line will be displayed.

The value is stored as part of the PROFILE options which are maintained individually by file type.

The COLS primary command will display the column-position (=COLS>) line at the top of the screen, which looks like:

```
=COLS> -----1-----2-----3-----4-----5
```

When there are TABS in effect, each tab column will be reflected in the COLS display as an underscore. Suppose you had Tabs set every 5 columns. When you displayed the COLS line, it would look like this:

```
=COLS> -----_-----1-----_-----2-----_-----3-----_-----4-----_-----5
```

Note that the COLS primary command was known as RULER in earlier versions of SPFLite. The current name conforms to ISPF conventions. The RULER command name is no longer supported.

COMPRESS - Compress Duplicate Strings

Syntax

```
COMPRESS      string-1  [ string-2 ]
                  [ start-column [ end-column ] ]
                  [ line-control-range ]
                  [ color-selection-criteria]
                  [ color-change-request]
                  [ X | NX ]
                  [ U | NU ]
                  [ MX | DX ]
                  [ ALL ]
                  [ TOP ]
```

Operands

string-1	The string being searched for. The string should contain the single occurrence of the value you wish to compress, unless you also specify string-2. See Description for issues related to Text, Picture and Regular-Expression search strings.
string-2	The string that replaces string-1. String-2 must be shorter than string-1.
start-column	Left column of a range (with end-column) within which the from-string value must be found. If no end-column operand, then the from-string operand must be found starting in start-col.
end-column	Right column of a range (with end-column) within which the from-string value must be found.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the from-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
color-change-request	A request for selection based on the highlight color of the from-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be examined, NU requests only non-User lines are to be examined. If neither U or NU are

specified, all lines in the range will be examined.

MX	MX requests that all lines which DO contain from-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain from-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status.
ALL	ALL requests that all lines in the line range will be compressed. If the ALL option is omitted, then by default the first line in the lines in the line range will be compressed. Note that this default action is what is done in other SPFLite commands when a NEXT operand is used. However, be aware that COMPRESS does not support any of the options FIRST, LAST, PREV or NEXT.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

COMPRESS can also be spelled as **CP**

Description

COMPRESS is used for two purposes: (1) you can specify a single string, to collapse repetitive spans of a search string to single occurrences of that string, and (2) you can specify two strings to repetitively change a longer string to a shorter one. The first action is called string compressing, and the second is called string condensing.

String Compressing

Every span of strings is compressed on each line of the selected line range if you say ALL; otherwise, just the first available line in the line range is compressed. For example, to compress spans of multiple * characters on the first two lines of a file, you could issue a command like this:

Command > COMPRESS '*' ALL U

U2 001 *** ABC *** DEF ***
000002 *** PDQ *** XYZ ***
000003 *** 123 *** 456 ***

Result:

000001|* ABC * DEF *
000002|* PDQ * XYZ *
000003 *** 123 *** 456 ***

String Condensing

Condensing text means to repetitively change one string into a shorter string. There are no restrictions on the values you choose to condense text, except that the replacement text in string-2 must be shorter than string-1. String-2 can be a zero-length string, so that all occurrences of string-1 are removed.

SPFLite condenses a line by scanning it for any occurrence of string-1, and when found, replaces it with string-2. If any replacements were made, the line is scanned *again* to see if any additional occurrences of string-1 are found, and these are again replaced with string-2. This continues until no more replacements can be made.

This is best explained by an example. Suppose you had a line with words having a closing parenthesis with space between them, and you wanted to remove the extra space, and further assume that the spacing before the parentheses are unequal. Let's say the line looked this this:

000001 (ABC) (DEF)

Now, you *could* issue many CHANGE commands like `CHANGE ')' ')' ALL` to remove all the extra blanks, which would require a lot of typing. Or, you could do a simple `COMPRESS ' '` but that would compress ALL blanks, not just the blanks before the parentheses, and also would not remove all blanks between the word and the right parentheses. Instead, let's use COMPRESS with the second string operand. What we need is `COMPRESS ')' ')' ').` This causes ALL instances of `')'` to be replaced with `')'.` Result:

000001 (ABC) (DEF)

Considerations when using non-specific search strings in COMPRESS

The COMPRESS search string can be any SPFLite string type, including T, P and R, and untyped strings when CASE T is in effect. When strings of such types are used, it is possible that more than one kind of character will match the search string. For example, `T'a'` will match on `'a'` or `'A'`, and `P'#'` will match digits `'0'` through `'9'`. Because these types of strings do not match one value alone they are called *non-specific strings*. When COMPRESS is provided with a non-specific string to search for, it finds the first actual string that matches the non-specific search item, and that actual value is used to compress any further repetitions of itself.

For example, given a line that contained `111222333`, a command of `COMPRESS P' #' ALL` will produce `123`. Even though all nine digits in `111222333` match against `P' #'`, in a FIND or CHANGE command, only identical spans of digits get compressed.

The reason that `111222333` does not compress down to a single digit is that COMPRESS first matches the non-specific string `P' #'` to the actual string `'1'`, and from that point on, only `'1'` characters are matched and compressed. The first instance of `'2'` starts a new string, and then the non-specific string `P' #'` matches the actual string `'2'` and from that point on, only `'2'` characters are matched and compressed, and so on. This same principle applies to Text strings, Pictures and Regular Expression strings.

This rule may seem a little complicated, but it's only reasonable to do it this way. After all, if COMPRESS really did reduce a string like `111222333` to a single digit, which one would it pick? 1, 2 or 3? Or something else? The way it does it as described above is the only way that makes sense. If you *really* wanted to collapse strings like `111222333` to a single digit, the best way to do it would be to first issue a command like `CHANGE P' #' '9' ALL` on the lines where you wanted to do this, then go back and `COMPRESS '9'` as needed.

COPY - Include an External File

Syntax

```
COPY [file-name | * ]
      [ from-line to-line ]
      [ { BEFORE | AFTER } line-label ]
```

Operands

file-name | * The name of the file you wish to include in the edit data.

If a single * is entered, it requests the filename to be fetched from the current clipboard contents.

from-line to-line Represents the starting line number, and ending line number, of the file you are copying from. If used, both the from-line and the to-line must be present, but need not be in ascending order. If one number is larger than the actual number of lines in the copied file, it is treated like the line number of the last line in the copied file was specified. If both numbers are larger than the actual number of lines in the copied file, the effect is the same as if an empty file were copied. When these numbers are omitted, the entire file is copied.

{ BEFORE | AFTER } If specified, the file is copied before, or after, a defined line label.

line-label If a before/after line label is not specified, the file is copied into the current edit file in one of the following ways:

- before a B line command
- after an A line command
- repeatedly before the lines in a BB block
- repeatedly after the lines in an AA block
- into a H/HH block, replacing the existing lines in that H/HH block

Description

COPY adds a copy of data that already exists to the edit session.

To copy data into an empty file:

On the Command line, type:

```
COPY filename
```

The filename operand is optional. If you do not specify it, the standard Windows file open dialog will be displayed. You may then select the filename in this dialog.

Press Enter. The data is copied.

To copy data into an existing edit file:

Use the [A](#), [B](#), [AA](#), [BB](#), or [H/HH](#) line commands to indicate where the copied data is to be inserted, or use the BEFORE/AFTER line-label option

- for the A or B command you may not enter a repeat value

- for the AA and BB command you are allowed to specify the **n**th line option (see the Help for AA and BB line commands for details)
- For the H/HH commands, mark off the line existing lines you want replaced by the file you are copying in
- To copy the file before or after a label instead of using the A/B line commands, define a label and use BEFORE label or AFTER label.

On the Command line, type:

COPY filename

The filename operand is optional. If you do not specify it the standard Windows File Open dialog will be displayed. You may then select the filename in this dialog.

Press Enter. The data is copied.

Note: If the current edit session is not empty, and you do not specify a destination, you will be asked to provide one of the destination options described above.

Determining the Default Directory

When COPY is specified with no filename operand at all, or with just a simple unqualified filename, the default directory is used for searching for the file. Or, if an Open dialog is displayed, Open dialog's starting directory will be determined as follows:

If there is a named file being edited in the tab where the command is issued, then the path for named file is used as the default for the command.

If there is no active file being edited, when the tab header displays (Empty):

- If File Manager is active, the currently displayed directory of File Manager will be used.
- If File Manager is not active, then the **General Options** setting for the **Default File Open Directory** drop-down will be used, either **Last Used Dir** or **Working Dir**.

Keep in mind that the “Working directory” means the directory that was active when SPFLite was started, if you start it from a command line. If you start SPFLite from a desktop icon, the working directory is the Start path defined in the icon's Property window.

CREATE - Create an External File

Syntax

```
CREATE      { line-control-range }
            [ filename ]
            [ X | NX ]
            [ U | NU ]
```

Operands

line-control-range The range of lines which are to be included by the command. Line control ranges provide a powerful tool to customize the range of lines to be included. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

Note: Prior editions of this document showed syntax that implied that the line-control range on **CREATE** was optional. That is not correct. You must specify a line-control-range, **unless** the line-control-range is implied via a C/CC or M/MM block. We are trying to emphasize that fact by showing the syntax in **red** above, and in braces rather than brackets.

file-name The name of the file you wish to create.

X | NX Specifies a subset of the line range to be included. **X** requests only excluded lines are to be included, **NX** requests only non-excluded lines are to be included. If neither **X** or **NX** are specified, all lines in the range will be included.

U | NU Specifies a subset of the line range to be included. **U** requests only User lines are to be included, **NU** requests only non-User lines are to be included. If neither **U** or **NU** are specified, all lines in the range will be included.

Abbreviations and Aliases

CREATE can also be spelled as **CRE**

Description

CREATE will use all or a specified portion of the Edit data to create a new external file. If you do not specify a full pathname as part of file-name, then the file will be created in the same directory as the file being edited. Note: If the filename already exists, then use the [REPLACE](#) command instead. The **CREATE** and [REPLACE](#) commands are almost identical, except that **CREATE** prevents overwriting of an existing file, whereas [REPLACE](#) does not.

To copy the entire Edit data into a new file:

On the Command line, type:

```
CREATE file-name .ZF .ZI
```

The filename operand is optional. If you do not specify it, the standard Windows File Open dialog will be displayed. Enter the filename in this dialog.

Press Enter. The entire Edit data is copied to the new filename.

To copy only a portion of the Edit data into a new File:

On the Command line, type:

```
CREATE filename line-control-range
```

The filename operand is optional. If you do not specify it the standard Windows File Open dialog will be displayed. Enter the filename in this dialog.

The line-control-range would typically specify the starting and ending line numbers to be included in the new file (or some other line-control-range variation). As described under ["Line Control Range Specification"](#) a variety of other possibilities for specifying the line range exist including marking the lines with the [CC/CC](#) or [MM/MM](#) lines commands.

Press Enter. The specified Edit data is copied/moved to the new filename.

File Format

If you wish to write a file in a format other than the normal default specified by your **PROFILE EOL** and **SOURCE** settings see ["Handling Non-Windows Text Files"](#) for additional info.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the **CREATE** command, the default directory used for writing the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the path for that active file is used as the default for the command.
- If there is **no** active file [when the tab header displays (Empty)], the currently displayed directory of File Manager will be used.

CRETRIEV - Conditional Retrieve

Syntax

```
CRETRIEV
```

Operands

None

Description

The CRETRIEV will move the cursor to the Home position, or will work as the RETRIEVE command, depending on the current location of the cursor.

If the cursor is currently on the command line, the CRETRIEV command will act like the [RETRIEVE](#) command and bring back to the Command line the previously issued command for use to modify and re-enter. Successive CRETRIEV commands will retrieve the previous, next previous, etc. commands. The Editor keeps the last 10 unique command lines available for retrieval. Note that the list of previous commands will be saved and recalled between SPFLite sessions.

If the cursor is not currently on the command line, then CRETRIEV acts like the Home key by placing the cursor at the beginning of the primary command area.

See also the [Options - General](#) setting for Minimum Retrieve Length.

CUT - Cut Data to the Clipboard

Syntax

```
CUT [ name ]
      [ line-control-range ]
      [ X | NX ]
      [ U | NU ]
      [ DELETE | DEL ]
      [ REPLACE | APPEND ]
      [ RAW ]
```

Operands

name	The name of a Named Private Clipboard. If omitted, the standard Windows clipboard is used.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
DELETE DEL	Requests that the lines selected for CUT be deleted following the operation.
REPLACE	The data selected by CUT will be placed in the clipboard and will replace any existing data already present. As noted, REPLACE is the default, and is provided for ISPF compatibility. Most users will simply omit REPLACE .
APPEND	The data selected by CUT will be added to the end of any existing data already in the clipboard.

Description

CUT saves copies of lines from an edit session to the Windows clipboard, or to a Named Private Clipboard, for later retrieval by the [PASTE](#) command or any other Windows application which handles Text-formatted clipboard entries.

The optional **APPEND** operand indicates that you wish the selected data to be **appended** to the current clipboard contents. Without the **APPEND** operand the clipboard is cleared of any current contents before adding the selected lines.

See also information on CLIP mode, and Named Private Clipboards, in [Windows Clipboard, Cut and Paste](#).

Note: The **CUT** command normally **copies** data to the clipboard (except when **M/MM** blocks are used, or the **DELETE** operand; see below). **In general, there is no subsequent deletion of text**, as is commonly done in the "cut" function of many word-processing programs. The command name **CUT** adheres to the IBM ISPF naming convention for this command, even if it is a bit "inaccurate" of a term. IBM decided on the command name **CUT** for ISPF a long time ago, before "cut" and "copy" had the more precise, commonly understood meanings they have today.

The **CUT** command will do both cutting and copying actions. If the line range is defined by a **C/CC** block, or by any type of line-control-range operand on the primary command line, a **copy** operation takes place. If the line range is defined by an **M/MM** block, or the **DELETE/DEL** operand is specified, a **cut** operation takes place.

Bear in mind that the **COPY** primary command is used for copying outside files into the current edit session, and has nothing to do with the clipboard. To copy from the clipboard, the **PASTE** command is required.

Specifying the Data to CUT

Two methods of specifying the data are possible.

- The first is the 'classic' SPF method of specifying line numbers as operands to the **CUT** command. You have the full range of options allowed by SPFLite's extended [line-control-range operands](#).
- Or, the lines may be marked by [C/CC](#) or [MMM](#) line commands. Depending on whether the **X** or **NX** operands are specified, the data **CUT** will be the **entire** contents of the specified line range if these operands are omitted, or the specified subset (**X** or **NX**) if the operands **are** specified.

The lines are copied from the edit session to the clipboard when the [C](#) or [CC](#) line commands are used, and are "moved" from the edit session to the clipboard when the [M](#) or [MM](#) line commands are used (or the **DELETE** operand). **Moving** the lines means that the data on those lines is copied to the clipboard, and then those lines are deleted.

As noted above, using **M/MM** automatically performs a true "cut" operation that deletes data afterwards, while defining your line range any other way, such as by using a **C/CC** block or a line-control-range command-line operand, performs a "copy to the clipboard" operation, analogous to how a Ctrl-C key works in most text editors. A **DELETE** operand will however **always** cause the selected lines to be deleted.

Using Raw Mode Copying

You can copy data in what is termed Raw Mode. Raw mode copy means that there are no End of Line terminators inserted into the copied text. So, when you copy text that encompasses more than one line, all the text from all of the lines are effectively glued together in one long character string. This action happens under the following cases:

- **CUT** primary command with **RAW** keyword
- The [\(CopyRaw\)](#) keyboard function
- The [\(CopyPasteRaw\)](#) keyboard function

When you copy text in Raw Mode, you can then go outside of SPFLite and paste the text thus copied as if were a single line, for example, into a Notepad session. By doing this, you would be able to perform a type of conversion from the kind of individual lines that SPFLite edits to text that may wrap across multiple lines, as editors such as Notepad commonly deal with.

Mapping the CUT and PASTE primary commands

Because the **CUT** and **PASTE** primary commands perform a **line-oriented** copying and pasting of data, similar to how **Ctrl-C** and **Ctrl-V** perform a **text-oriented** copying and pasting of data, many users will want to create

key mappings for the CUT and PASTE commands.

In order to be of the most benefit, you may wish to define these mappings so that the current cursor location is saved and restored, so that its location doesn't "jump around" as SPFLite processes these commands.

Here are some suggested key mappings using **Alt-C** and **Alt-V** you may wish to try:

Copying lines of text to the clipboard: Use C/CC or M/MM to define the line range first:

Map **Alt-C** to: **(SaveCursor) (Home) [CUT] (Enter) (RestoreCursor)**

Pasting lines of text from the keyboard: The key mapping inserts an Aline command for you on the current line:

Map **Alt-V** to: **(SaveCursor) (LineNo) [A] (Home) [Paste] (Enter) (RestoreCursor)**

DELETE - Delete Selected Lines

Syntax

DELETE	DUP [line-control-range]
DELETE	{ NOTE xNOTE zNOTE } [line-control-range]
DELETE	[string [PREFIX SUFFIX WORD <u>CHAR</u>]] [start-column [end-column]] [line-control-range] [color-selection-criteria] [X NX] [U NU] [ALL FIRST LAST NEXT PREV] [TOP]

Operands

There are three basic modes of **DELETE** operation. **DEL DUP** is single purposed to delete adjacent duplicate lines. **DEL NOTE** is used to delete NOTE lines within a specified range while the third provides a more general use delete function that can delete data lines that match a wide range of matching capabilities.

Note that **DELETE DUP** and **DELETE NOTE** do not permit other **DELETE** keywords to be used. In particular, you **cannot** issue a command like **DELETE DUP ALL** or **DELETE NOTE ALL**. Within the specified line-control range, or within the entire edit file, **all** duplicates or **all** notes are deleted; **the "ALL" is implied but cannot be used on these commands.**

Note that to distinguish these special command modes, if the words **DUP** or **NOTE** are to be used as **strings** in the general purpose delete function, they must be enclosed in quotes to prevent them being treated as keywords.

DUP Specifies this command is to delete adjacent duplicate lines from the line range. The keyword **DUP** is reserved, and may **not** be specified as **DUPS**, **DUPLICATE** or **DUPLICATES**. When determining if two lines are duplicates of each other, SPFLite checks that they are of identical length and are byte-for-byte identical for the entire length of the lines, including any leading and trailing blanks that may exist.

Note that the duplicate-removal processing of **SORT UNIQ** is **not the same as** **DELETE DUP**. Refer to the [SORT](#) command for more information.

NOTE **NOTE** specifies this command is to delete all **=NOTE>** lines from the line range.

xNOTE **xNOTE** specifies this command is to delete all extended **xNOTE>** lines from the line range, where 'x' is any letter from **A** to **Y**. To delete all extended notes of type **A**, issue **DELETE ANOTE**.

zNOTE **zNOTE** specifies this command is to delete all extended **xNOTE>** lines **of all types** from the line range, where 'x' is any letter from **A** to **Y**. **DELETE ZNOTE** will **not**

delete "plain" **=NOTE>** lines. As a reminder, you cannot create extended **xNOTE** lines of type **Z**; that is, **ZNOTE** is not allowed as a line command.

The keyword **NOTE** and the forms **ANOTE** through **ZNOTE** are reserved. **NOTE** may **not** be specified as **NOTES**.

string

If **string** is present, a string-based delete is requested; only lines containing **string** are eligible for deletion within the selected line range. If neither **ALL**, **FIRST**, **LAST**, **PREV** or **NEXT** is specified on a string-based **DELETE**, **NEXT** is assumed. Note that a "string-based delete" means the entire line having the string is deleted; it does not mean to delete the string alone from the line. The string may be unquoted, or simply quoted, or may be any of the standard string types **C**, **T**, **X**, **P** or **R**.

If **string** is omitted, a line-based delete is requested; lines within the selected line range are deleted. If neither **ALL**, **FIRST**, **LAST**, **PREV** or **NEXT** is specified on a non-string-based **DELETE**, **ALL** is assumed.

CHARS | WORD | PREFIX | SUFFIX

These options are allowed only for string-based deletes; they describe the "string context" in the same way that a **FIND** or **CHANGE** command does.

If omitted for a string-based delete, the string will be searched for in **WORD** mode if **C** **W** or **T** **W** appears in the middle of the status line, and in **CHARS** mode if **C** or **W** appears in the middle of the status line. **WORD** mode or **CHARS** mode can be set by **FIND WORD** or **FIND CHARs**, respectively.

ALL

For line-based deletes, **ALL** is allowed mainly for compatibility with prior releases. **ALL** is assumed if not specified, except for string-based deletes which assume **NEXT**. Note that to delete the entire file, **DELETE ALL** is permitted. When deleting the entire file, the command **DELETE** must be fully spelled out and cannot be abbreviated as **DEL**, and the **ALL** keyword is not assumed but must be explicitly specified. These rules help prevent accidental deletion of the file. A lone **DELETE** with no other options is illegal.

line-control-range

The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

start-column

Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.

end-column

Right column of a range (with start-column) within which the search-string value must be found.

color-selection-criteria

A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "[Color Selection Criteria Specification](#)". Refer to that section of the documentation for details.

X | NX

Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be deleted, **NX** requests only non-excluded lines are to be deleted. If neither **X** or **NX** are specified, all lines in the range will be eligible to be deleted.

U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
FIRST or NEXT	Starts at the top of the specified range and searches ahead to find the first occurrence in the specified line-control-range and delete that line. NEXT is assumed for string based deletes.
LAST or PREV	Starts at the bottom of the specified range and searches backward to find the last occurrence in the specified line-control-range and delete that line.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

DELETE can also be spelled as **DEL**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

DELETE removes lines from an edit session.

Specifying the Data to Delete

Two methods of specifying the data are possible.

- The first is the classic ISPF method of specifying line numbers as operands to the **DELETE** command. You have the full range of options allowed by SPFLite's extended line-control-range operands.
- Or, the lines may be marked by **C/CC** line commands.

Note carefully:

- With SPFLite, **DELETE ALL** is a legal command, and it will delete the entire file. There are **NO** "are you sure" safeguards. If you delete your entire file by mistake, immediately issue the command **UNDO** to recover your file. Or, you may issue the **CANCEL** command to prevent loss of data, but you will also lose any pending unsaved changes as well.
- When issuing a **DELETE ALL** command, you must **fully spell out** the command name **DELETE**. If you try to specify **DEL ALL**, the command will be rejected. Since most users, by habit, type the shortest command they can, this provides a degree of protection against typing this command by mistake.

Depending on whether the **X** or **NX** operands are specified, the data deleted will be the entire contents of the specified line range if these operands are omitted, or the specified subset (**X** or **NX**) if the operands are specified.

When performing a string-based delete, the options **ALL**, **FIRST**, **LAST**, **PREV** and **NEXT** are all available.

When performing a line-based delete, the **ALL** option is assumed if none of **ALL**, **FIRST**, **LAST**, **PREV** or **NEXT** are specified. However, to perform a list-based delete using **FIRST**, **LAST**, **PREV** or **NEXT**, some type of 'line qualification' must also be used. For example, you cannot just say **DELETE FIRST**, otherwise you will get the message, "No search string, but no ALL specified". What you need for a valid line qualification to do a line-based delete using **FIRST**, **LAST**, **PREV** or **NEXT** is one of the following:

- A defined **C/CC** block
- A pair of defined line labels on the **DELETE** command
- A defined line tag on the **DELETE** command
- An **X** or **NX** option on the **DELETE** command

Repeatable **DELETE**

Just as **FIND** and **CHANGE** can be selectively repeated using **RFIND** (or **RLOCFIND**) and **RCHANGE**, so can the string-based version of **DELETE**. You can issue a command like **DELETE 'ABC'** and use **RFIND/RCHANGE** to selectively find and delete lines.

RFIND, **RLOCFIND** and **RCHANGE** now have support to allow this action to be performed following an initial **DELETE** command, when the **DELETE** has a find-string operand. As a reminder, **RFIND** (or **RLOCFIND**) is typically mapped to the F5 key, and **RCHANGE** to the F6 key.

For example, if you place **DELETE 'ABC'** on the command line, then press **F5** instead of **Enter**, **DELETE** will locate the next line containing **ABC**, but will not delete it. To actually "follow through" and delete that line, press **F6**. If you **don't** want to delete that line, you can press **F5** to find the next line with **ABC**, or simply do something else.

Other examples of the **DELETE** command

To delete all excluded lines:

```
DELETE X  
or  
DELETE ALL X
```

To delete all non-excluded lines:

```
DELETE NX  
or  
DELETE ALL NX
```

To delete a range of lines using line numbers:

```
DELETE 20 30  
or  
DELETE ALL 20 30
```

To delete a range of lines using line labels:

```
DELETE .HERE .THERE  
or  
DELETE ALL .HERE .THERE
```

To delete only excluded lines within a range of lines using line numbers:

```
DELETE X 25 35
      or
DELETE ALL X 25 35
```

To delete lines tagged with an :AB line tag:

```
DELETE :AB
      or
DELETE ALL :AB
```

To delete all lines containing the “ABC”:

```
DELETE ALL ABC
```

To delete all lines containing the “DEF” highlighted in RED:

```
DELETE ALL "DEF" RED
```

To delete all unexcluded lines containing the text “abc” as a prefix:

```
DELETE ALL T'abc' PREFIX NX
```

Note: **DELETE** cannot be used to delete zero-length blank lines based on a string. See [NDELETE](#) for an example of how to do this.

DIR - Display Folder in File Manager

Syntax

DIR	[file-pattern]
------------	-------------------------

Operands

file-pattern If specified, this provides the file pattern mask which you wish File Manager to use when displaying the directory. e.g. a string of `*.txt` would display only the .TXT files in the directory

Description

The **DIR** command will switch to File Manager and display the directory containing the current Edit / Browse file, using the File Pattern if specified..

If the current tab is not a normal Edit / Browse session (e.g. a CLIP or Set-Edit session) the command is rejected.

Note that the DIR command provides a function comparable to that provided by the Windows Explorer when the context-menu item called Open Containing Folder is selected for a given file name.

DO - Invoke a KB Command file

Syntax

DO	command-file
----	---------------------

Operands

command-file The name of the command-file to be executed. Do **not** specify the **.DO** file type. e.g. to execute MYJOB.DO simply enter **DO MYJOB**.

Description

The **DO** command will fetch the specified DO file and execute the contents.

See "[Keyboard Macros](#)" for more details.

DROP - Delete Specified Lines in a Tag Group

Syntax

DROP	line-tag
	[FIRST LAST ALL]

Operands

line-tag	The specific line tag identifying the line tag blocks which are to be processed by the command. Line tags are a method of marking possibly noncontiguous lines as belonging to a related 'group'. A full discussion of line tags and how to use them is in "Working with Line Tags" .
FIRST	only the first line in each tag block is deleted; all other lines in each tag block are kept
LAST	only the last line in each tag block is deleted; all other lines in each tag block are kept
ALL	all lines in each tag block are deleted; all other lines tagged with line-tag which are not part of a tag group (2 or more adjacent lines) will remain.

Description

The **DROP** command only pertains to "proper" tag-blocks, which are contiguous groups of 2 or more lines having the same line tag. When such a block is 'dropped', one or more lines are deleted from the block. When multiple tag blocks having the same tag exist, each tag block is processed independently. The **DROP** command applies its processing to every tag block in the entire edit file having the specified line tag.

A lone line having a tag which is not adjacent to another line having the same tag is not considered a proper tag-block, and will be ignored by the **DROP** command.

EDIT - Open a File for Editing

Syntax

```
EDIT      [ file-name | NEW | * ]  
          [ .Profile-name ]
```

Operands

file-name | The name of the file to be edited.

*

If you wish to edit a file whose actual name is NEW, the name must be enclosed in quotes as 'NEW'.

If a single * is entered, it requests the filename to be fetched from the current clipboard contents.

NEW The new file tab will be opened with a new, empty file.

.Profile-name This optional operand allows you to specify an overriding Profile to be used for the Edit session. It simply consists of the name of the desired Profile, preceded by a . (period)

Description

The **EDIT** command loads a file into the edit work space for editing. If no file-name operand is specified, then a standard Windows file open dialog will be presented for you to select the file for editing.

If you are currently editing a file in the Tab, the requested file (or **NEW**) will be opened in a new Tab.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the **EDIT** command, the default directory used for searching for the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the Path for **that** active file is used as the default for the command.
- If there is **no** active file [when the tab header displays (Empty)], the current displayed directory of File Manager will be used.

Overriding Profile

When you wish to use a different Profile from the one indicated by the file's extension, simply provide the alternate profile name, preceded by a period, on the command line. For example to Edit MYSOURCE.BAS using the TXT profile the command would be

```
EDIT MYSOURCE.BAS .TXT
```

The use of the alternate profile is for the duration of this session **only**, it does not alter any permanent Profile processing.

END - End the Edit Session

Syntax

```
END
```

Operands

None

Description

The **END** command may be used in an Edit or Browse session, in a Set Edit session, in a Clip Edit session, or in the File Manager.

Using END in an Edit session

To end an edit session by using **END**, do one of the following:

- Enter **END** on the Command line, or
- Press a keyboard key to which **END** is assigned; the default setting is F3; or
- Right-click on the file tab; this causes SPFLite to take the same action as the **END** command.

These methods will process the data as described below, and will close the current edit tab.

- Use the standard Windows close button **X** in the Title bar, or
- Enter **EXIT** on the Command line, or
- Enter **=X** on the Command line, or
- Press a keyboard key to which **EXIT** is assigned. There is no default setting for this; you would have to map a desired key using the KEYMAP facility if you wanted to have it available.

These methods will cause **END** processing to be performed as described below for **all** active tabs. SPFLite will then terminate after all files are closed.

The actions performed on the data being processed in each tab will be as follows:

- If there is no data present and the tab shows **(Empty)**, the tab is closed without being saved.
- If the tab is an unmodified Browse session, the Browse tab is simply closed. If the current data has been modified **and** the AUTOSAVE setting contains a PROMPT value (either **AUTOSAVE ON PROMPT** or **AUTOSAVE OFF PROMPT**) then a prompt will be issued to confirm the loss of the modified data. An exit from the **END** can be requested to allow saving the data before continuing.
- If the tab is a **CLIP Edit** session, the current data will be returned to the Windows Clipboard and then the Clipboard tab will be closed.
- If the tab is a **SET Edit** session, the current SET variables are saved in SPFLite's SET variable configuration file, and will be present the next time SPFLite is restarted.
- If data is present, and no changes were made to the data since it was first opened or since the last SAVE command, the tab is closed without being saved.

- If the data has unsaved changes, then the **AUTOSAVE** and **AUTOBKUP** settings of the current Profile will be applied. **AUTOSAVE** determines whether or not unsaved changes are automatically saved and whether or not you will be prompted before this action is taken. **AUTOBKUP** controls whether a backup of the current file will be automatically created. See the Help descriptions of **AUTOSAVE** and **AUTOBKUP** for more information.

When a file is automatically saved, when either the **END** command is issued or a file tab is clicked with the right mouse button, and **AUTOSAVE** is in effect, a message of **File Saved** will appear on the status line, as a reminder that the automatic save took place. This message will disappear as soon as any key is pressed. When a **Multi-Edit** session is ended, the File Saved message will contain a count of the number of files saved.

Using **END** in the File Manager

The **END** command in File Manager is used to 'return' to the next higher level in the file list being displayed. The particular type of display shown as a result of **END** depends on the kind of display currently shown and on how you navigated to that display originally. To the extent possible, **END** will try to 'undo' the navigation you previously did, in much the same that a **CHDIR ..** command does in a DOS prompt. The **END** command replaces the **DIR UP** functionality that was present in prior versions of SPFLite.

As with Edit sessions, if you right-click the File Manager tab it causes SPFLite to take the same action as the **END** command.

- If the current file display is a directory list, **END** will replace the current display with a display of the parent directory. Once you get to the root directory like **C:** the **END** command will have no effect and the display will remain at the root directory.
- If you are displaying one of the standard File Lists (**Recent Files**, **Favorite Files** or **Found Files**), **END** will replace the current File List display with a display of the last-displayed directory list.
- If you are displaying Named Favorites File Lists because you clicked on the **Named Favorites** entry and then on your own named list, **END** will replace the named File List display with a display of the Named Favorites list. If you clicked on the **Named Favorites** entry but did not click on one of your own named lists, **END** will replace the current File List display with a display of the last-displayed directory list.

ENUMWITH - Change Increment for Enumerate Functions

Syntax

ENUMWITH	[number]
-----------------	-------------------

Operands

number Specifies a new increment value. **number** may be specified either as a decimal value or as hex value in the form **X'hexnum'**. The value of **number** or **hexnum** cannot be zero.

If **number** is omitted, the current setting of **ENUMWITH** is displayed as a message.

Description

By default, the enumeration keyboard primitive functions ([Enum](#)), ([EnumHexLc](#)) and ([EnumHexUc](#)) use an increment value of **1**. You can set the increment value to any positive number by using the **ENUMWITH** primary command.

Once the **ENUMWITH** value is set, it applies to **every** SPFLite edit session until set again, not just the edit session it was issued from.

The **ENUMWITH** value is not persistent. If you close SPFLite and then restart it, the **ENUMWITH** value gets reset back to **1**.

Because **ENUMWITH** is a primary edit command, and the **(Enum*)** functions operate only in Power Typing mode, you cannot change the enumeration increment value while you are in the middle of Power Typing. If you need the increment value changed, you **must** issue the **ENUMWITH** command **before** beginning Power Typing mode.

As an Edit primary command, **ENUMWITH** cannot be issued from the File Manager.

EOL - Set End-Of-Line Handling

Syntax

EOL	[CRLF CR LF NL AUTO AUTONL hh hhhh NONE]
------------	--

Operands

CRLF	Use the carriage return / line feed combination as the line separator.
CR	Use the carriage return character as the line separator.
LF	Use the line feed character as the line separator.
NL	Use the new line character as the line separator.
AUTO	Automatic detection of line separators is performed, with lone CR characters used to overprint. See discussion below.
AUTONL	Automatic detection of line separators is performed, with lone CR characters treated as new lines. See discussion below.
hh	Where 'hh' may be any two valid hex characters. The resulting character will be used as the line separator.
hhhh	Where 'hhhh' may be any four valid hex characters. This two character string will be used as the line separator
NONE	There are no line separator characters used. The file must have an LRECL value greater than 0 which will be used to perform the line separation function

Description

The EOL setting is stored in the file type's profile and is used when reading and writing files to support handling different file formats.

The command itself has no default; you must specify a line-delimiter operand. SPFLite will assume a never-before seen file-type uses CRLF, the standard delimiter in Windows.

For additional information on these values and their effect, see ["Handling Non-Windows Text Files"](#).

The value is stored as part of the PROFILE options which are maintained individually by file type.

Handling of EOL AUTO and EOL AUTONL

The End of Line options EOL AUTO and AUTONL allow for automatic detection of line terminations, possibly containing inconsistent and spurious line terminators, so that files edited across different system, mainframe SYSOUT files, and other inconsistently-terminated text files can be opened, viewed and edited in a reasonable way.

EOL AUTO and AUTONL may be applied to non-mainframe files as well, to handle situations where a file's line

termination is inconsistent for some reason. A possible cause of this is a file shared between Windows and Unix on a network and edited with different editors applying different line endings.

Line terminations under EOLAUTO and AUTONL are handled as follows:

FF characters delimit lines, and cause a =PAGE> marker to be placed in the sequence area.

- Scrolling commands UP PAGE and DOWN PAGE will locate these marked lines.
- Since UP PAGE and DOWN PAGE will move the file to these =PAGE marker lines, which may have a variable number of lines involved, to regain the 'full screen motion' that UP/DOWN PAGE does in other files, you can use a scroll amount of HALF or DATA, or you could enter a numeric value for a specific number of lines. For most users who would have used UP/DOWN PAGE, UP/DOWN by the DATA scroll amount should work well for them.
- When you have a file that shows this =PAGE> marker on a line, and you PRINT this file, you will have a Form Feed sent to the printer for every line containing the =PAGE> marker.

A lone **LF** is treated as a line delimiter equivalent to CR,LF

“**Spurious**” **CR** characters that seemingly don't belong there, such as CR,CR,LF are ignored. For example, in the sequence CR,CR,LF, the first CR is spurious; the remaining CR,LF is a normal line termination.

A **CR,FF** or **CR,LF,FF** sequence is considered as the end of one line, followed by a page separator line.

A hex value of **X'1A'** (Ctrl-Z) at the end of the file is ignored.

Choosing between EOL AUTO and EOL AUTONL

A “lone CR” character – that is, a CR not followed by LF, FF or another CR – is sometimes produced by software that attempts to *overprint* the data in order to simulate underscores or bold print. It may also exist in non-Windows text files; older versions of Macintosh and some lesser-known systems used CR as a line termination.

Because of this, a lone CR character might be used for two different, conflicting purposes.

To handle this, you may choose between EOLAUTO and EOLAUTONL. These two options work as follows:

For all but the “lone CR” situation, EOLAUTO and EOLAUTONL work identically.

Action taken for EOL AUTONL: CR = New Line

When EOL is set to AUTONL and SPFLite detects a lone CR in a file, the CR is considered to be a “new line” and is treated as if the lone CR were actually a normal CR/LF line termination.

Action taken for EOL AUTO: CR = Overprint

When EOL is set to AUTO and SPFLite detects a lone CR in a file, it is considered to be an overprint request. At this point, SPFLite will buffer the lines involved in the overprint request until a ‘normal’ line terminator is found, and then attempts to simulate an overprint. This means that, on a column-by-column basis, it examines the characters that are attempting to ‘occupy’ the same column at the same time. For each two characters involved in this way:

- When a blank and a non-blank character are in the same column, the non-blank character ‘wins out’.
- When two non-blank characters are in the same column, and they are identical, it is recognized as a “bold font” type of overprint, and is not a problem; the non-blank character is retained.

- When two non-blank characters are in the same column, but one of the non-blank characters is an underscore, the non-underscore character 'wins out'. That way, the underscore character will never 'obliterate' the meaningful data.
- When two non-blank characters are in the same column, and they are not identical, and neither is an underscore, an "overprint clash" has occurred. When this happens, the first such character is retained and any others are discarded. Where SPFLite detects this, it will issue a warning message. If you frequently see this warning, it suggests the file having lone CR characters was not written to overprint, but either has foreign line delimiters or is perhaps 'damaged' in some way. The way to address this is to change the EOL setting from EOLAUTO to EOLAUTONL.

Note for Hercules users

Hercules users are familiar with a utility called **HercPrt**, which (among other things) has the ability to take a SYSOUT file and format it into a PDF file that looks remarkably like a computer printout on "green bar" paper - complete with sprocket holes and perforation! This is cute and very clever, but it also uses a large amount of disk space. By using alternating background colors (along with EOLAUTO and PAGE ON mode), it is possible to very closely simulate the effect of a HercPrt-formatted file without the PDF disk-space overhead. You will still be editing or browsing ordinary text files in native mode, but the display will have the look and feel of paging through an actual hard copy printout.

If you wish to match the same colors generated by the HercPrt utility, make the main background color white, and the alternative background color a light green. A good starting point for a HercPrt-like light green color you could try is BRG value 233, 255, 233. You may wish to tie the profile attributes to a specific file extension like SYSOUT.

EXCLUDE - Exclude Lines (Edit Mode)

Syntax

```
EXCLUDE      [ search-string ]
                [ start-column [ end-column ] ]
                [ FIRST | LAST | NEXT | PREV | ALL ]
                [ PREFIX | SUFFIX | WORD | CHAR ]
                [ line-control-range ]
                [ U | NU ]
                [ color-selection-ccriteria ]
                [ TOP ]
```

Operands

search-string	The search string that identifies the lines to be excluded
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with end-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of search-string.
NEXT	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
CHAR	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line

[Control Range Specification](#)". Refer to that section of the documentation for details.

U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

EXCLUDE can also be spelled as **X**, **EX**, **EXC**, or **EXCLUDED**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the EXCLUDE command to find a search string, and exclude (conceal) the lines that contains the string from the display. Note that the normal selection options of **X** and **NX** are not allowed, since EXCLUDE operates **only** on non-excluded lines. See "[Working with Excluded Lines](#)" for more information.

To exclude the next non-excluded line that contains the letters ELSE without specifying any other qualifications:

On the Command line, type:

EXCLUDE ELSE

Press Enter. Since no other qualifications were specified, the letters ELSE can be:

- Uppercase or a mixture of uppercase and lowercase.
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To exclude the next line that contains the letters ELSE, but only if the letters are uppercase:

On the Command line, type:

EXCLUDE C"ELSE"

and press Enter.

This type of exclusion is called a character string exclusion (note the C that precedes the search string) because it excludes the next line that contains the letters ELSE only if the letters are found in uppercase. However, since no other qualifications were specified, the exclusion occurs no matter where the letters are found on a non-excluded line, as outlined in the previous list.

Note: EXCLUDE cannot be used to exclude blank lines. See [NEXCLUDE](#) for an example of how to do this.

See [Working with Excluded Lines](#) for more information.

EXCLUDE - Exclude Lines (File Manager Mode)

Syntax

EXCLUDE	File-pattern
----------------	---------------------

Operands

File-Pattern The filter pattern used to select files to be Excluded.

Note that if multiple masks are specified, they must be enclosed in quotes if you are using the default command separator of ;. e.g. **EXCLUDE "A*.TXT;B*.TXT"**

Abbreviations and Aliases

EXCLUDE can also be spelled as **X**, **EX**, **EXC**, or **EXCLUDED**

Description

You can use the EXCLUDE command to eliminate files from a File List display. Since displays showing File Lists do not support a File Patterns input field (since File Lists internally support patterns on each individual entry in a File List) this command provides another level of filtering.

The File-pattern operand is exactly the same format and structure as the File Patterns option field

The **File-pattern** operand should contain a list of one or more file patterns or “wildcards” of files you wish to exclude. The format used is the based on that used by Windows Explorer or the command line. (A ? question mark matches any single character, and an * asterisk matches zero or more characters up to the next period or delimiter.) To exclude all .BAS files in a directory, specify ***.BAS**. To exclude all .BAS and .TXT files, specify ***.BAS;*.TXT**

Note that when multiple masks are specified, they must be enclosed in quotes if you are using the default command separator of ;. e.g. **EXCLUDE "A*.TXT;B*.TXT"**

EXIT - Terminate SPFLite Session

Syntax

```
EXIT | =X      [ NOREOPEN ]
              [ END | CANCEL [ DELETE | PURGE ] ]
```

Operands

NOREOPEN	If NOREOPEN is specified, SPFLite will not save the list of currently open Edit tabs for use at the next normal SPFLite start. This means the next SPFLite startup will begin in the File Manager tab, with no other tabs being automatically loaded.
	Whether SPFLite re-opens files at start-up is controlled by a preferences selection on the Options -> General screen. The NOREOPEN option will cause the list of currently-open files to be discarded, even if the Re-open checkbox is enabled. (The files themselves are not discarded - only the list that refers to them is discarded.)
<u>END</u>	If specified (or defaulted) it requests a normal END command be issued for each edit tab as it is closed. Normal AUTOSAVE processing will be performed.
CANCEL	If CANCEL is specified in place of END, it requests a CANCEL command be issued in each edit tab as it is closed. If a tab contains modified, unsaved data, you will not be notified, your changes will be discarded.
DELETE	If specified along with the CANCEL operand, then as well as canceling the edit session, the file being edited will be deleted from the file system.
PURGE	If specified along with the CANCEL operand, then as well as canceling the edit session, the file being edited will be permanently deleted from the file system. i.e. it will not be sent to the recycle bin

Description

The **EXIT** command with no operands will terminate all currently open Tabs and is equivalent to clicking the upper-right **X** box on the SPFLite window's title bar.

It effectively performs an **END** or **CANCEL** command on each active tab. If **END** is issued, the **AUTOSAVE** processing for each tab takes place as usual.

The command **=X** is an abbreviation for **EXIT**. The **=** is required, since **X** by itself is the short form of **EXCLUDE**.

Note: In IBM ISPF, the notation **=X** is used to accomplish a "quick exit". IBM terms this notation a "jump function". In ISPF, jump functions are used to navigate within the hierarchy of Dialog Manager panels. SPFLite does not have a Dialog Manager, and the **=X** notation here is the only supported ISPF-compatible jump-function syntax, provided as a convenience.

The command **=X** happens to use the same syntax as the Set Symbol notation. Because of this, a Set Symbol

named **X** cannot be used for any purpose. If a set symbol of **X** is created in the SET variable screen, the value is saved but cannot be used.

FAVORITE - Add Current File To A Favorite List

Syntax

```
FAVORITE [ Favorite | list-name ]
```

Operands

list-name

Specifies the name of a **Named Favorites** File List. The **list-name** operand is treated as case-insensitive, but will be recorded as-typed-in, the first time a particular file list name is used.

When **list-name** is omitted, the name of the file currently being edited or browsed is added to the standard file list, which is called the **Favorite Files** File List.

The **list-name** operand cannot specify any of the predefined File List names, which are **Recent**, **Paths**, **Fav**, **Favorite**, **Favourite**, **Open** and **Found**. The various spellings of **Fav**, **Favorite** and **Favourite** all mean the same **Favorite Files** File List, which is the same list that is used if no list-name is specified.

Abbreviations and Aliases

FAVORITE can also be spelled as **FAV** or **FAVOURITE**

See also [Using AUTOFAV to add to File Lists](#)

Description

The **FAVORITE** primary command is used to add the name of the current edit file or browse file to the **Favorite Files** File List or a **Named Favorites** File List.

If currently editing or browsing the file **ABC.TXT** in path **C:\MYPATH**, then the command

FAV

will result in the file name **C:\MYPATH\ABC.TXT** being added to the **Favorite Files** File List.

The command

FAV Mylist

will result in the file name **C:\MYPATH\ABC.TXT** being added to **Mylist.FLIST**.

The same file name can be added to as many File Lists as desired. It is not an error to add a file to a File List when the file name is already present. (If you do this, you will see a message saying that the file had already been added previously to the list.)

The **FAVORITE** command cannot be issued from Clipboard edit sessions, or when editing the **SET** variable list.

In the File Manager, the **FAV** primary command is not allowed, but the **A** line command is available. When used, the **A** line command works the same as **FAV** without the *list-name* operand; that is, it adds the file name into the standard **Favorite Files** list.

To remove a name already in a File List, the **Forget** line command **F** can be used from File Manager. To delete an entire File List, the **Delete** command **D** can be used from File Manager. The **Open Files**, **Recent Files**, **Recent Paths**, and **Named Favorites** entries cannot be deleted.

Reserved FILELIST names

The File List names **FOUND**, **OPEN** and **PATHS** are reserved for displaying the special lists described in [RECALL](#). This is in addition to **RECENT**, **FAV**, **FAVORITE** and **FAVOURITE** which were already reserved.

Reserved File List names cannot be used for user-defined named favorites for the **FAVORITE** and [MAKELIST](#) commands. Even though you can issue a command like **RECALL OPEN**, you cannot say **FAVORITE OPEN**, because that would imply you were creating (or adding a file name to) a **Named Favorites** File List called **OPEN**, which SPFLite has reserved for its internal list of currently-open files.

FF - Find In Files

Syntax

```
FF           string
              [ start-column [ end-column ] ]
              [ CHARS | WORD | PREFIX | SUFFIX ]
              [ NF ]
```

Operands

string	Any string value accepted in an edit session is permitted here. The <i>string</i> may be unquoted, or quoted without a string type, or may be quoted with a string type of C , T , X , P or R .
	Unquoted strings, or string quoted without a string type, will be treated as either C or T , depending on the CASE C/T code that appears on the SPFLite status line. The CASE C/T code for the File Manager can be changed by the CASE command , the same as in an Edit or Browse session.
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
CHARS WORD PREFIX SUFFIX	Specifies the 'search context' for the string, the same as is done for the FIND command in the editor.
	If not specified, a CHARS search is done if the status line shows C or T , and a WORD search is done if the status line shows C W or T W . These two defaults can be set by the FIND CHARS and FIND WORDS commands, as in an edit tab.
NF	If the NF option is used, a search is made for files that do not have the <i>string</i> present on any line.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Operational Note Regarding the File Patterns Field

When the **Find in Files** command **FF** is issued in the File Manager tab, the **current** File Patterns mask is used to select one or more file patterns to search for. Due to the timing of how SPFLite processes the File Patterns field, **the file mask you wish to use must already be in effect** when the **FF** command is issued.

In practice, this means that **you cannot change the File Patterns field AND issue the FF command at the**

same time. If you attempt to do that, **SPFLite will use the last-known File Patterns field instead of the one you just entered**. That means you could end up searching files you didn't intend to search.

The correct way to handle this situation is to (a) modify the File Patterns field as desired; (b) Press Enter; (c) type the desired FF command and press Enter a second time.

This issue only applies to when the File Patterns entry field is visibly displayed, which will be on a standard directory display. If you issue a Find in Files command **FF** on some other list, such as the Recent list or a user-defined FLIST, a File Patterns field is not available, and the search is then limited to the files displayed.

Description

The **Find in Files** command **FF** is issued in the File Manager tab. It references each file in the currently displayed directory list or File List. For each file in the displayed list, the **Find in Files** command **FF** will look for the string in the file, in the same way that a **FIND** command looks for strings in an edit file, applying string types **C**, **T**, **X**, **P** or **R** and search types **CHARS**, **WORD**, **PREFIX** or **SUFFIX** likewise in the same way.

If the string is found anywhere in the file (or, if the **NF** option is used and the string is **not** found anywhere in the file), the file is remembered as meeting the File in Files search criteria. Once all the files in the original list have been examined, the **Find in Files** command **FF** creates the **Found Files** File List, and the files in this **Found Files** File List become the current list of files displayed in File Manager.

You can then use the **Found Files** File List to search again, refining your list by searching for additional strings, or you can use the **Found Files** File List to enter File Manager line commands as usual.

If you wish to edit all files together containing a certain string, you can use the **Find in Files** command **FF** to locate them, and use the Multi-Edit line command **M** on each file to begin a multi-edit session using all the files you found in the search process. You can also use the [File Manager ALL command](#) to edit or Multi-edit every file listed in a File List.

When you issue a Find in Files command **FF** and then open the **Found Files** File List, the **FF** command, and all options that were specified with it, will appear on the top line of the display, like this:

FILELIST > Found Files: FF T'ABC'

This will help in keeping track of and remembering what was being searched for, especially in cases where the **FF** command might have been done previously, and the Found Files FILELIST is being redisplayed (perhaps long) after the time it was created.

You can find additional information and examples in [Performing Searches with Find In Files](#).

Note: The Find in Files command **FF** should not be confused with the new **FF** alias for the edit primary command **FIND**; the two commands are not related. However, if you issue a File Manager command of **FF ABC**, and then open a file listed in the Found Files FILELIST, you can quickly find the string ABC by retrieving the **FF** command (usually by the command mapped to F12). Assuming there is no outstanding **CC** or **MM** blocks in the edit file, the **FF ABC** in the edit session will find the string ABC within the file, the same way that a regular FIND edit command would. This can be a time-saving shortcut.

FIND - Find a Character String

Syntax

FIND	WORDS CHARS context)	(Alter search
FIND Alias - FF (See Line Range / Pending Copy Conflict discussion below.)	search-string FIND) [start-column [end-column]] [FIRST LAST NEXT PREV ALL] [PREFIX SUFFIX WORD CHAR] [LEFT RIGHT] [line-control-range] [color-selection-criteria] [color-change-request] [X NX] [U NU] [MX DX] [TOP]	(Perform a normal

Operands

There are two basic formats for the **FIND** command as shown in the syntax boxes above. The first is used to alter the basic search context as it relates to use of the WORD option in normal searching. This is covered below under the heading "Search Context".

The following operand descriptions are for the normal use of the **FIND** command.

search-string The search string that identifies the lines to be found

start-column Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.

end-column Right column of a range (with start-column) within which the search-string value must be found.

FIRST Starts at the top of the data and searches ahead to find the first occurrence of search-string.

LAST Starts at the bottom of the data and searches backward to find the last occurrence of search-string.

NEXT Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. **NEXT** is the default.

PREV Starts at the current cursor location and searches backward to find the previous occurrence of search-string.

ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
LEFT	LEFT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the left-most occurrence of search-string is found, and any other instances on that same line are ignored.
RIGHT	RIGHT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the right-most occurrence of search-string is found, and any other instances on that same line are ignored.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-Word characters.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
	See also the Line Range / Pending Copy Conflict discussion below.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
color-change-request	A request for highlighting of the found string. The full syntax and allowable operands for the color-change-request are discussed in " Color Change Request Specification ". Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be examined. U requests only User lines are to be examined, NU requests only non-User lines are to be examined. If neither U or NU are specified, all lines in the range will be examined.
MX	MX requests that all lines which DO contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is

Abbreviations and Aliases

FIND can also be spelled as **F**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the **FIND** command to locate line(s) within the file which contain a specified string.

To find the next occurrence of the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

FIND ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming that **CASE T** is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word).
- In either an excluded or a non excluded line.
- Anywhere within the current boundaries.

To find the next occurrence of the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

FIND C'ELSE'

Press Enter. This type of search is called a character string search (note the **C** that precedes the search string) because it finds the next occurrence of the letters **ELSE** only if the letters are in uppercase. However, since no other qualifications were specified, the letters can be found anywhere in the file, as outlined in the preceding list.

For more information, including other types of search strings, see [Finding and Changing Data](#) and [Specifying a Picture or Format String](#).

Note: When the **FIND** search operand is a Regular Expression (a string with an **R** type code) and reverse-order searching is done with **PREV** or **LAST**, only the left-most occurrence on any given line is found. That is, the command

FIND R'ABC' PREV

is treated as if it were specified as

FIND LEFT R'ABC' PREV

and

FIND R'ABC' LAST

is treated as if it were specified as

FIND LEFT R'ABC' LAST

Note: FIND cannot be used to find zero-length blank lines based on a string. See [NFIND](#) for an example of how to do this.

Setting the search context with FIND WORDS and FIND CHARS

Operands

WORD		WORDS		
CHAR		CHARS		

Note: WORD and WORDS are interchangeable, as are CHAR and CHARS.

Description

This variation of the **FIND** command is a special extension to the syntax of the **FIND** command, and must be specified **exactly as shown**, with no other regular **FIND** operands.

FIND WORDS is used to set the default search context to **WORD** mode.

FIND CHARS is used to set the default search context to **CHARS** mode.

This means that whenever a **FIND**, **CHANGE** or similar command is used, and none of the operands **CHARS**, **WORD**, **PREFIX** or **SUFFIX** are specified, the **FIND** or **CHANGE** command will proceed using the default assigned here.

This is convenient, for example, when many **FIND** and **CHANGE** commands are required that need **WORD** mode. By setting the search context to **WORD** mode, the operand **WORD** can be omitted from the individual **FIND** and **CHANGE** commands, making them shorter and faster to type.

CHARS mode is the installation default, and is how ISPF normally operates. The main reason to issue **FIND CHARS** would be to change back to the standard active after a **FIND WORDS** command had been issued.

Setting the search context in this manner is only a default, applied when none of the operands **CHARS**, **WORD**, **PREFIX** or **SUFFIX** are specified. Any of these keywords can be used on a **FIND** or **CHANGE** to override the current default search context, whatever it may be at that time.

When the default search context is set to **WORDS**, a **W** will appear the **C/T** indicator on the status line that shows the current **CASE C/T** mode. Thus, depending on the **CASE** mode, the indicator will show either **C W** or **T W** on the status line.

If you issue a **RESET** command with no operands, the default search context will revert to the system-wide default (either **WORDS** or **CHARS**) that is defined on the Options - General screen using the checkbox, [Use WORD as the default for FIND/CHANGE commands](#). If that default is **CHARS** (checkbox not checked) the indicator will revert to either **C** or **T** on the status line.

The search context checkbox defined on the [Options - General](#) screen is the default when a file is opened. You can set the search option to **FIND CHARS** or **FIND WORDS** any time you wish on any file tab. This setting is not saved in the profile.

Line Range / Pending Copy Conflict - a Play in Three Acts

Synopsis

- When there is a pending **C/CC** or **M/MM** block, **FIND** will keep the block pending, so you can use **FIND**, **RFIND** and/or scrolling commands to locate a place for the pending lines to be copied into. This understanding provides SPFLite with ISPF compatibility, and is the way **FIND** with pending **C/CC** blocks is commonly used on the mainframe.

- When there is a pending **C/CC** or **M/MM** block, a new alias of FIND called **FF** will use the block to define the range of lines to be searched for, instead of keeping the lines pending. This is how **FIND** used to work before a recent change was made to it. (This new alias **FF** is not the File Manager command of the same name. See the final note at the end of this section for more information.)

This new arrangement resolves a long-running conflict that pitted IBM ISPF compatibility with our desire for SPFLite to provide as many productivity extensions as possible. We appreciate everyone's patience as we worked through this issue and "threaded the needle the right way".

It is quite a tale, and here it is, told in three acts ...

Act I

The extended line-range-operand support provided by SPFLite allowed you to specify a line range that a primary command is to use, either by specific command line operands (like **.FROM .TO** or **.20 .250**), or by marking the line range with **C/CC** line commands.

This extended line-range-operand support was a **general** feature that applied to **all** primary commands that took a line range, including **FIND**. Therein lies the rub ...

Act II

It was found that a common use case in ISPF was to first define a block of lines using **C/CC**, and then use the **FIND** command to locate a destination for those lines to be copied into, a task usually involving repetitive **FIND** commands (via **RFIND**) and/or manual scrolling. By SPFLite making the general assumption (for **every** primary command, including **FIND**) that a **CC** or **MM** block **always** defined a line range, that assumption prevented this commonly-practiced ISPF use case from being performed.

Here's an example of this problem:

Suppose you want to copy line **2500** to some other part of the file, a long way from line 2500. You're not sure **where** the right place **is** to copy the line, but you do know there's a string of **ABC** somewhere near where the new line needs to go. Also, there's more than one ABC in the file, and so just finding **one** of them is not enough; you have to find the **right** ABC.

Incidentally, there is no string of **ABC** on line 2500 itself.

So, you **wanted** to do this:

Scene I

- Enter a **C** line command on line 2500
- Go to the top of file, with an **UP MAX**, or maybe with a mapped key like Alt Page-Up or Ctrl Page-Up
- Enter on the command line, **FIND "ABC"** to find the next ABC in the file
- You **might** find the right ABC, or you might **not**. You may have to keep searching, probably with **RFIND** (mapped to the F5 key) or maybe by manually scrolling up and down.
- All this time, there a message on the screen, "**Pending Destination line range command**", because the **C** is waiting for you to decide where you want to put an **A** or **B** somewhere, to finish what the **C** started.
- Finally, you make the decision, pick a line, put an **A** command on it, press Enter, and the line is copied.

Well, that's what you **wanted** to do, and in IBM ISPF, that's how it always worked.

However, because of the general feature (added in "Act I"), **every** primary command that took a line range (including **FIND**), treated a pending **C/CC** or **M/MM** block as defining the range of lines that the primary command (**FIND**, in this case) was to operate on.

Thus, what **actually** happened was this:

Scene II

- Enter a **C** line command on line 2500
- Go to the top of file, with an UP MAX or maybe with a mapped key like Alt Page-Up or Ctrl Page-Up
- Enter on the command line, **FIND "ABC"** to find the next ABC in the file
- Because there is a pending **C** command on line 2500, **FIND** decides that it is supposed to **only look at line 2500 !**
- Line 2500, the only line **FIND** inspected, does **not** have a string of ABC - not that you wanted to look at line 2500 anyway
- **FIND** finds **nothing**, and reports, "Bottom of data reached".

Hmm ... that's not right.

What happened in Scene II is not ISPF compatible, and doesn't do anything useful. So, the **FIND** command was recently **exempted** from the general line-range-defining role of **C/CC** and **M/MM**, in the name of ISPF compatibility, to make it possible to **FIND** with an active/pending **C/CC** block present. Because of that exemption, **FIND** with a pending **C/CC** or **M/MM** works like Scene I shows it to work.

Act III

The exemption for **FIND** that was added in Act II is great, and solves a big problem. However, that exemption for **FIND** removed the convenience of SPFLite's ability of **FIND** to use **C/CC** and **M/MM** as general command line-ranges, which is also an important and valuable use case - too useful to give up.

For example, suppose you wanted to quickly count the number of left parentheses in the first 100 lines of the file.

You would **like** to be able to do something this:

Scene III

- Put a line command of **C** on line 100. This applies the **C** command from line 100 back to line 1
- Issue the command **FIND '(' ALL**
- You would **like** the command to issue a message like, **CHARS '(' found 45 times**

But, that won't work, because of the exemption added in Act II. The block of lines is just "left hanging", and you get a message, **"Pending Destination line command range"**. So, it doesn't do anything **wrong** - it just doesn't do anything **period**.

To restore that capability, a **FIND** command alias of **FF** has been defined.

What **FF** does is to treat any pending **C/CC** or **M/MM** line commands as defining the find operation's line range to search for, rather than keeping those lines as "pending" while scrolling through the data. So, the new **FF** command works like the old **FIND** command did, before the **FIND** exemption was put in. That's because **FF** uses pending **C/CC** or **M/MM** line commands in the "general" way that other SPFLite primary commands do, and not like the "exempted **FIND**" command does.

In other words, the following now **will work**:

Scene IV

- Put a line command of **C** on line 100. This applies the **C** command from line 100 back to line 1
- Issue the command **FF '(' ALL**
- The command **will** issue a message like, **CHARS '(' found 45 times**

Other than the different name, and the different way that **C/CC** and **M/MM** blocks are handled, **FF** and **FIND** are

the same command and accept exactly the same parameters.

Note: The Find in Files command **FF** should not be confused with the new **FF** alias for the edit primary command **FIND**; the two commands are not related. However, if you issue a File Manager command of **FF ABC**, and then open a file listed in the Found Files FILELIST, you can quickly find the string ABC by retrieving the **FF** command (usually by the command mapped to F12). Assuming there is no outstanding **CC** or **MM** blocks in the edit file, the **FF ABC** in the edit session will find the string ABC within the file, the same way that a regular FIND edit command would. This can be a time-saving shortcut.

FIND - Find File Name in File Manager List

Syntax

FIND	search-string
-------------	----------------------

Operands

The following operand description applies only to the FIND in File Manager command. This is **NOT** the normal edit FIND command; for that command, see "[Find a character string](#)" for more information.

search-string The search string that identifies the file names to be found. If the search string contains blanks, it should be enclosed in quotes. Note that **none** of the special literal types (P, T, C, X etc.) may be used nor can any of the search modifiers such as **WORD**, **PREFIX**, **SUFFIX** etc. be used. This is a simple string match search.

Description

You can use the File Manager **FIND** command to locate specific filenames within a File Manager directory list or File List. The search will look for the search-string anywhere within the path and/or filename of the displayed list. When found, the located line will be scrolled to the top of the File Manager screen.

Note: You may use the [RFIND](#) or [RLOCFIND](#) commands to continue the search for the next occurrence of the search-string.

FLIP - Reverse Exclusion Status of Lines

Syntax

```
FLIP          [ search-string ]
              [ start-column [ end-column ] ]
              [ FIRST | LAST | NEXT | PREV | ALL ]
              [ PREFIX | SUFFIX | WORD | CHAR ]
              [ line-control-range ]
              [ U | NU ]
              [ color-selection-criteria ]
              [ TOP ]
```

Operands

search-string	The search string that identifies the lines to be flipped
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of search-string.
<u>NEXT</u>	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax

and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

All commands that affect the exclusion status of lines are "unified" with a common design patterned after the **EXCLUDE** command. That makes all of the commands very powerful. In ISPF, the **FLIP** command always applies to all lines by default, but in SPFLite you must now use **ALL** if you want to affect all lines.

You can use the **FLIP** command to find a search string, and invert the visibility state of the line on which it is found. i.e. exclude the line if currently visible, or show the line if currently excluded. Note that the normal selection options of **X** and **NX** are not allowed, since **FLIP** must process both types of lines. You may also wish to review "[Working with Excluded Lines](#)" for more information.

To **FLIP** the status of the next line that contains the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

FLIP ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming CASE T is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To **FLIP** the status of the next line that contains the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

FLIP C"ELSE"

and press Enter.

This type of search is called a character string search. Note the **C** that precedes the search string. The command flips the next line that contains the letters **ELSE**, but only if the letters are found in uppercase. Since no other qualifications were specified, the exclusion change occurs no matter whether the letters are found on an excluded or non-excluded line.

FOLD - Display in Uppercase

Syntax

FOLD	[<u>ON</u> OFF]
-------------	----------------------------

Operands

ON | OFF The desired FOLD status

Description

FOLD allows an edit or browse session to display data only in upper case, even if the data itself contains lower case letters. This can be useful in viewing mainframe SYSOOUT files, where lower case is used less frequently, and having lower case data displayed can be a distraction at times.

If **ON** is specified, SPFLite will display all text in upper case. The text itself is not altered. FOLD affects only the displayed text data on the screen.

If **OFF** is specified, then text will be displayed normally (upper and lower case).

The value is stored as part of the PROFILE options which are maintained individually by file type.

GLUEWITH - Specify Join String for GLUE operations

Syntax

GLUEWITH	[string]
-----------------	-------------------

Operands

string

The *string* operand is optional. It may be specified like a **FIND** string, which can be either plain text or a quoted string if it contains embedded blanks or special characters.

When **GLUEWITH** is specified without a *string* value, the current **GLUEWITH** setting is displayed as a message.

When **GLUEWITH** is specified with a *string* value, the current **GLUEWITH** setting is replaced. It becomes effective in every current edit session, and in future edit sessions until changed again, because it is a persistent value.

Description

By default, the Glue line commands **G/GG** and **TG/TGG** concatenate glued lines together with no intervening blanks or other characters. The **GLUEWITH** command can be used to define a string to be inserted between such glued lines.

The **GLUEWITH** string is a global value, and has an installation default of "" (a zero-length string). This value is stored in the SPFLite INI file, and is thus persistent. The only way to view or modify the **GLUEWITH** string is by using the **GLUEWITH** primary command in an edit session.

The **GLUEWITH** string setting applies only to the Glue commands **G/GG** and **TG/TGG**, **not** to the Join commands **J/JJ** and **TJ/TJJ**.

When a **GLUEWITH** string is in effect (other than a zero-length string) and a **G/GG** and **TG/TGG** line command is used, you will see a message displayed showing **Glued with "xxx"** where **xxx** is the **GLUEWITH** string in effect.

This message is a reminder that the **GLUEWITH** string was defined, in case you set it earlier and perhaps forgot that it was still defined. If you didn't intend for this **GLUEWITH** string to be used, you can **UNDO** the glue operation, change the string to something else or to a null string with **GLUEWITH ""** and then try your glue command again.

Because **GLUEWITH** is a less-frequently used command, there is no abbreviation for it. If desired, a **SET** symbol could be defined to abbreviate this. For example, suppose you wanted to use the abbreviation **G** for **GLUEWITH**. Then, you wished to set the **GLUEWITH** string to an asterisk. The following commands would accomplish this. Note that **SET** symbols are also persistent, so once the symbol **G** were defined, it would remain defined for future use.

```
SET G = GLUEWITH
=G '*'
```

See [G / GG - Glue Lines Together](#) and [TG / TGG - Text Glue Lines](#) for examples of using **GLUEWITH**.

HELP - Display Online Help

Syntax

HELP	[help-topic]
-------------	-----------------------

Operands

help-topic If this operand is provided, the SPFLite help file will be opened to the information for that particular topic. If the operand is omitted, then the Help file will be opened at the Introduction.

Description

This command provides quick access to the SPFLite Help information.

For **help topic**, you may enter any of the primary command names, or any of the line control commands. For example, to learn how to use the SORT command, issue **HELP SORT**.

In addition to internal command names as operands, the HELP command recognizes the following:

HELP MACROS to bring up the SPFLite Macro support Help file

HELP THINBASIC to bring up the thinBasic Help file
or
HELP BASIC

HEX - Enable Hex Display of Data

Syntax

HEX	[<u>ON</u> OFF]
------------	----------------------------

Operands

ON | OFF The desired hex display status

Description

The **HEX** command is used to switch the edit screen between the normal Character only display, to a **HEX** display which shows the Hex characters representing the normal character in a vertical orientation below the normal character.

In **HEX** mode, you may alter data either by entering characters on the 'normal' character line as usual, or you may enter any valid HEX characters (0-9, a-f and A-F) in the vertical positions below the normal character line.

The value is stored as part of the PROFILE options which are maintained individually by file type.

Note that while in **HEX** display mode, functions to cut and paste data are not enabled. If you wish to cut and paste data from a file currently displayed in **HEX** mode, you must set **HEX OFF** first, **then** perform the cut or paste operation.

HIDE - Hide Excluded Lines

Syntax

HIDE	[<u>ON</u> OFF]
------	---------------------

Operands

ON | OFF The desired Hide status

Description

The **HIDE** command will turn on **HIDE Mode**. Excluded lines are normally collapsed to an "excluded-line placeholder" line indicating how many lines are represented. In **HIDE** mode, the excluded-line placeholder is removed from the display. The data lines themselves are still present in the edit file.

Unlike ISPF, **HIDE** does **not** accept the **HIDE X** form, because only excluded lines can be hidden. **HIDE** or **HIDE ON** are equivalent to ISPF's **HIDE X**.

To indicate the presence of these hidden excluded lines, the line number field of the preceding line will be Underlined. Other than this, there is no visible indication that HIDE Mode is in effect. In particular, there is no status-line indicator for HIDE Mode.

For this display to work properly, the fixed-width font you choose for editing must be capable of underlining text in a readable way. Most fonts have this property.

Just as with non-hidden excluded lines, line movement commands spanning these lines will include them in the requested action. Likewise, when line commands use the line-number form (like **C34** instead of a **CC/CC** pair) then the hidden/excluded line counts as one.

Note that **FIND**, **CHANGE** and similar commands cannot be performed in line-by-line mode through the "interior" of a hidden excluded range. If you need to use such commands on hidden data, you must either unhide it with a **HIDE OFF** command, or else you must use the **ALL** option on **FIND**, **CHANGE** and similar commands.

See [Working with Excluded Lines](#) for more information.

HILITE - Control Text Highlighting Options

Syntax

```
HILITE      {
            [ ON | OFF ]
            [ AUTO ]  [ FIND ]
}
```

Operands

ON	Turn on the specified AUTO and/or FIND options. If only ON is specified it will be treated as ON AUTO FIND .
OFF	Turn off the specified AUTO and/or FIND options. If only OFF is specified it will be treated as OFF AUTO FIND .
AUTO	If specified with an ON/OFF operand, then the AUTO option is set accordingly. If specified without an ON/OFF operand, it is treated as ON AUTO . The AUTO option specifies whether the edit text should be colorized based on the contents of the .AUTO colorize control file. See "Colorize Files" for full details of colorize support.
FIND	If specified with an ON/OFF operand, then the FIND option is set accordingly. If specified without an ON/OFF operand, it is treated as ON FIND . The FIND option specifies whether the result of a FIND or CHANGE command should be highlighted on the edit screen when found/changed.

Description

The **HILITE** command controls the setting of two SPFLite Profile options.

- AUTO** The **AUTO** setting controls whether colorization support for the file type should be activated. This support, even with **AUTO ON**, still requires a colorize control file to properly activate. See [Automatic Colorization Files](#) for full details of colorization support.
- FIND** The **FIND** setting controls whether the result of **FIND** / **CHANGE** commands should be highlighted in the text or whether simply moving the cursor to the location is sufficient.

The value is stored as part of the PROFILE options which are maintained individually by file type.

Note that all **HILITE** keywords are optional, but at least one must be chosen; **HILITE** by itself is not a valid command.

Be aware that the colors controlled by the **HILITE** command and by automatic colorization have nothing to do with "highlighting pens". These are completely different features. See [Working with Virtual Highlighting Pens](#) for more information.

JOIN - Join lines Using Find/Change Strings

Contents of Article

[Syntax](#)

[Operands](#)

[Description](#)

[Performing joins when a search Picture of `P'\[\]` or `P'\[\]` is desired](#)

[Simulating a JOIN with a search string of `P'\[\]` or `P'\[\]`](#)

[Simulating a JOIN with a search string of `P'\[\]`](#)

[Suppose you really want to JOIN and not just simulate it?](#)

[Performing joins more complex than JOIN can support](#)

[Line joining and line exclusion](#)

[Simulating CHANGE operands that are not available on JOIN](#)

Syntax

```
JOIN      P'from-string'  |  R'from-string'
          [ to-string ]
          [ FIRST | LAST | NEXT | PREV | ALL ]
          [ PREFIX | SUFFIX | WORD | CHAR ]
          [ line-control-range ]
          [ color-selection-criteria ]
          [ X | NX ]
          [ U | NU ]
          [ TOP ]
```

Operands

from-string The search string you want to look for. The from-string must be defined as either a P-type Picture string or an R-type RegEx string, with a very restricted format that is discussed below.

to-string The string you want to replace from-string. This may be any standard *change string* including P-type Picture strings and F-type Format strings. The to-string is optional, and if omitted, it is treated the same as `P'!!'`. That is, when to-string is omitted, the replacement string is the same as the value found by the from-string. Note that the alignment Picture codes of [and] do not correspond to data values, and so these are not part of the value represented by `P'!!'`.

FIRST Starts at the top of the data and searches ahead to find the first occurrence of from-string.

LAST Starts at the bottom of the data and searches backward to find the last occurrence of from-string. **See Release Note below.**

NEXT Starts at the first position after the current cursor location and searches ahead to find the next occurrence of from-string. **NEXT** is the default.

PREV Starts at the current cursor location and searches backward to find the previous occurrence of from-string. **See Release Note below.**

ALL	Starts at the top of the data and searches ahead to find all occurrences of from-string.
PREFIX	Locates from-string at the beginning of a word.
WORD	Locates from-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
CHAR	Locates from-string regardless of what precedes or follows it.
SUFFIX	Locates from-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the from-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Release Note for LAST and PREV keywords

The **SPLIT** and **JOIN** commands are a new technology feature; their development is quite involved, and is ongoing. As of the writing of this Help documentation, the **LAST** and **PREV** keywords on **JOIN** may not work correctly or may not be supported at all, initially. As this situation changes, we will keep you updated via the SPFLite web site for any new capabilities that are made available.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**
SUFFIX can also be spelled as **SUF** or **SFX**
WORDS can also be spelled as **WORD**
CHARS can also be spelled as **CHAR**

Description

The **JOIN** edit primary command is used to selectively combine lines of text based on a search string. After a join occurs, a line on which the **from-string** is found will be combined either with the line before it, or with the line after it. So, each time a join takes place, two lines of text will become one line of text.

Notes:

- The **SPLIT** command performs the opposite function, breaking one line into two. See [SPLIT - Split Lines Using Find/Change Strings](#) for more information.
- The "joining" action could be termed a "physical" join, without trimming of leading or trailing blanks from any of the lines involved in the **JOIN** process. In that sense, the action is comparable to what is done by the **G** and **GG** "glue" line commands, rather than by the **J/JJ** "join" line commands or the by text-mode line commands **TG/TGG** and **TJ/TJJ**.
- The effect of inserting text between lines that takes place with **J/JJ** and **TJ/TJJ** commands, or with the **GLUEWITH** command, can be achieved by how the **to-string** is defined. In that sense, the capabilities of the **JOIN** primary command are more powerful and flexible.
- **SPLIT** and **JOIN**, as new SPFLite technologies, have been extensively tested, but it is possible you may run into issues using them. If you have any questions about how these commands operate, feel free to leave us your feedback on the SPFLite web site.

The **from-string** describes the *join point* where the joining operation takes place. Because a join operation joins a given line to the line that precedes it or to the line that follows it, a join can take place at only one of two join points: at the beginning or a line, or at the end of a line.

The **from-string** may be specified as a Regular Expression, in addition to a Picture string.

The **from-string** must be either:

- a Picture string in one of the following formats:

P' [string]

JOIN is being asked to perform a *left-join operation*. The value of **string** must appear on the left side of lines within the line range in order to be joined; otherwise any lines not beginning with **string** will be ignored for **JOIN** purposes. When a **[** Picture code appears in the **from-string** of a **JOIN** command, it must appear in the left-most position of the Picture string, and nowhere else.

P' string]

JOIN is being asked to perform a *right-join operation*. The value of **string** must appear on the right side of lines within the line range in order to be joined; otherwise any lines not ending with **string** will be ignored for **JOIN** purposes. When a **]** Picture code appears in the **from-string** of a **JOIN** command, it must appear in the right-most position of the Picture string, and nowhere else.

- a Regular Expression string in one of the following formats:

R' ^expression'

JOIN is being asked to perform a *left-join operation*. The regular expression **must** start with the **^** directive to indicate the left-hand edge of the line. The remaining expression may be any valid RegEx expression.

R' expression\$'

JOIN is being asked to perform a *right-join operation*. The regular expression **must** end with the **\$** directive to indicate the right-hand edge of the line. The remaining expression may be any valid RegEx expression.

It is not possible to directly join one line to **both** the line that precedes it **and** the line that follows it, at the same time. That is to say, a **from-string** of a **JOIN** command cannot be specified in the form of **P' [string]'** or **R' ^expression\$'**. For any given line, only one **JOIN**, on one side of a line, is allowed. **JOIN** cannot perform a left-join and a right-join at the same time, since this would imply converting three lines of data into one line in a single join operation, an action that SPFLite does not support. See the discussion below for ways to simulate such complex joins.

Note: Because the **from-string** must be a Picture or RegEx, there could be cases where you are trying to

find data containing characters that are already defined as special-purpose Picture or RegEx codes. If you need such characters treated as ordinary data rather than as Picture codes, you can escape those characters by preceding them with a \ backslash. See [Specifying a Picture or Format String](#) for more information. See [Specifying a Regular Expression](#) for escaping within a Regular Expression.

Note: In a Picture string, any character can be escaped with a backslash, whereas in a Regular Expression, only a limited number of "special" codes can be escaped. If you try to escape a Regular Expression character that is not eligible for being escaped, your Regular Expression will not properly match strings in the way you expected. This behavior is caused by the design of the Regular Expression "engine" and is outside the control of SPFLite. Read the Regular Expression documentation carefully on this point.

The **from-string** must be in exactly one of these two formats. The **string** value cannot be omitted. That is, you cannot have a JOIN **from-string** appearing literally as `P' ['` or `P'] '`. The main reason for this is that the [and] codes represent edges of a line, but do not represent actual data values. So, a picture of `P' ['` or `P'] '` would actually represent a zero-length string, and the string-search engine in SPFLite will not "find" strings of zero length, because there is literally nothing to find. For the same reason, a picture of `P' [] '` is not allowed either. See the discussion below for ways to simulate such joins.

The **to-string** is optional. If you omit it, the found-string is copied as is (ignoring the [or] Picture code), the same as if a **to-string** of `P' ! '` was specified. That means the following JOIN commands all work the same way:

```
JOIN P' [ABC'  'ABC'  
JOIN P' [ABC'  P' !'  
JOIN P' [ABC'
```

It is not illegal to ask **JOIN** to left-join line 1 of a file, or to right-join the last line of a file. Since there is no data to join such lines to, the **JOIN** request is simply ignored for those lines.

Note: While the **to-string** can be a Format string, you will find that using a Picture change string here should address most of your **JOIN** requirements when the **to-string** isn't a "simple" change string. Where a Format change string comes in handy is when the **from-string** is a Picture (as is always true for **JOIN**), and you need one or more codes of = in the **to-string** that don't match the corresponding character positions in the **from-string**. See [Specifying a Picture or Format String](#) for more information. When you have an editing requirement of this kind, you will get an error message if an = code is misplaced in a Picture change string; that is your 'clue' that you need a Format string instead. (These comments about the use of the = codes in Pictures also apply to the related < > and ~ codes, which operate in a similar way and have similar rules.)

See [Working with SPLIT and JOIN Commands](#) for example usage of the **JOIN** command.

Performing joins when a search Picture of `P' ['` or `P'] '` is desired

As noted above, **JOIN** will not accept a search Picture of `P' ['` or `P'] '` or `P' [] '`. These strings are illegal in **JOIN**, because they all represent zero-length strings, and `P' [] '` (if legal) would represent a zero-length line; the string search-engine in SPFLite will not find such zero-length strings or lines, because there is literally nothing to find. Suppose, though, you had some reason doing such joins anyway. How could you go about it?

Simulating a JOIN with a search string of `P' ['` or `P'] '`

A join of this type implies that you want to join a line to another line, such that the contents of the line itself are not the deciding factor. You probably are not (intentionally) trying to find zero-length lines, but (a) you don't care about the exact contents of the line, (b) you are almost certainly using F5 and F6 (or other keys mapped to **RFIND/RLOCFIND** and **RCHANGE**) to selectively join lines because you have to manually inspect each one to decide whether to JOIN it or not, and (c) **in case** you do run into a zero-length line while doing this, you don't want to be stopped from what you are planning on doing.

Suppose you wanted to selectively find lines, perhaps with the **FIND** command or by some other means, and

when you find such lines (even zero-length lines), you want to join them to the line before or after.

Because the line command **G** will perform a "physical join" of the line it is on to the line that follows it, it basically performs a function similar to what a right-joining command of **JOIN P'] ' '** would do, if such a command were legal. Let's say you issued some kind of **FIND** command, and you keep pressing F5 to **RFIND** the desired line(s). You now want to right join the line - **whatever** it contains, **even** a zero-length line - to the line after it. For convenience, let's map the line command **G** to the **Ctrl Shift F5** key. The KEYMAP string would appear as **{G}**. Then, when you find a desired line, you would right-join it by pressing **Ctrl Shift F5**.

Performing a function similar to what a left-joining command of **JOIN P' [' '** would work basically the same way. However, since the join/glue line commands always perform operations that amount to joining on the **right**, you would have to turn a right join into a **left** join by moving the cursor up one line and then issuing a the **G** line command. Let's map the cursor movement and the line command **G** to the **Ctrl Alt F5** key. The KEYMAP string would appear as **(Up) {G}**. Then, when you find a desired line, you would right-join it by pressing **Ctrl Alt F5**. So, instead of joining the "current" line to the "previous" line, we move the cursor up one line, and then join the "previous" line to the "current" one - but the result is the correct one.

This technique will also work when one of the lines being joined is a zero-length line. In such cases, the zero-length line is effectively deleted.

Simulating a JOIN with a search string of **P' [] '**

A join of this type implies that you want to find lines of zero length and "join" them to an adjacent line. However, by definition, since the line is of zero-length, "joining" such a line - whether to a preceding line or to a subsequent line - is the same thing as deleting it. So, you are actually trying to delete zero-length lines by using a **JOIN** to do it.

You cannot directly "find" strings of zero-length, because as noted above, SPFLite's string search engine will not find zero-length strings, as there is literally nothing to find. However, you **can** find **lines** in which no characters are present. This relies on the **NFIND** command. If you issue a command of the form:

NFIND P'='

you are asking SPFLite to find lines in which not even a single character (of length **one**) matching a Picture of **P'='** is found. Since a Picture of **P'='** represents **any** character, the **NFIND** command finds any lines where there are no characters found at all, because the test for finding characters that match the pattern of **P'='** has failed. The only lines that can possibly meet this condition are lines of zero length. This command is repeatable by using the **RFIND** or **RLOCFIND** command (traditionally mapped to F5).

Once you find the desired zero-length lines, just delete them. You could map a key to the **{D}** delete line command for this purpose, or perhaps you might wish to "mark" such zero-length lines with a special character (like the **?** character described below) and then go back and delete all of them with a primary command such as:

DELETE '?' ALL

If your goal was simply to delete all zero-length lines from a file, the easiest way to do this is as follows, which relies on the **NEXCLUDE** command, abbreviated as **NX**. (This is not the only to do it, but it's a good general example.)

RESET
NX P'=' ALL
DELETE ALL X

Suppose you really want to JOIN and not just simulate it ?

The main problem to doing this is dealing with zero-length lines. The easiest way to overcome this is to put some data on those lines so they aren't zero-length any more. One method of doing this is to **APPEND** a blank to such

lines. Appending a blank to a zero-length line will make it a line of length 1. For most **JOIN** purposes, that should be good enough. Here is a sequence that will 'fix' all the zero-length lines in a file:

```
RESET
NX P'= ALL
APPEND ' ' ALL X
```

The [Pad to Length command PL](#) can take a / or \ modifier. Putting **PL/** on line 1 of a file will ensure that every line of the file is at least one character long.

Performing joins more complex than JOIN can support

You may encounter cases where the **JOIN** command won't do everything you want. You might want to join a line to the line before it **and** to the line after it, at the same time. You may have text already colored by highlighting pens and you need precise control over how the text colors are affected by joining. You may need special features supported by **CHANGE**, such as **TRUNC**, **MX**, **DX**, etc. These and other cases may require a different approach.

Keeping in mind that a **JOIN** is a type of "change" to a line, you can perform more-complex joining by doing this in two stages. First, use a **CHANGE** command to insert "user-defined join points" into your data, and then go back and use one or more **JOIN** commands to actually combine them. This technique has the nice feature that after the first part, you can go and manually inspect all of the user-defined join points you just put in, and verify they are all where you want them, possibly adding and removing a few before the second part, if you have certain special cases where some join points have to be taken out and others added. Because the **CHANGE** command, and any manual editing of your own, will have placed these user-defined join points exactly where you need them, only a simple form of **JOIN** will be required to combine the lines.

To do this, you might want to map some special ANSI character that you rarely use in your own data, and use that to represent your user-defined join points. If necessary, you can use the [\(Ansi\)](#) function to get any ANSI character into the clipboard, and then use it as a value for **KEYMAP**.

Suppose you had a string "AB-CD-EF" appearing between lines .ONE and .TWO. In some cases, AB-CD-EF is the only thing on a line, neither preceded nor followed by any characters (even spaces), and in other cases it may appear on some lines next to other data. You want to take the lines where AB-CD-EF is the only thing on a line, and join that line **both** to the line before it **and** to the line after it, at the same time. You can't do that directly with **JOIN**, but (assuming that ? does not appear in your data) you could do the following instead. The idea here is that you would use the ? character to temporarily mark a user-defined join point, which you go back and process with subsequent **JOIN** commands, which also remove the temporary join marks. The same mark is used on both sides, so that when you do the **JOIN** commands, you will be sure that you just change lines in which AB-CD-EF is the only thing on a line, rather than in places where it happens to just begin or end a line. That is, you want to make sure you join the correct lines, and nothing else.

Note here that **CHANGE** can have a find-picture containing both [and], whereas **JOIN** allows only one of these codes (but not both). We use a change Picture of **P[?]?!** in the **CHANGE** command for conciseness. You could have also used a simple string of **?AB-CD-EF?!** instead. Note how the **JOIN** command change-strings effectively delete the ? character by not including it in the characters appearing in the change-string.

As you can see, the to-string of the **JOIN** command can be a zero-length (null) string. The net effect is that the ? character defines where the join takes place, and then that ? character is removed by being replaced with ..

If you want to do this, you have to explicitly specify an empty **to-string** of .., because if you just omit the second string operand, it's assumed to be **P!..!**, which *copies* the original found-string rather than *deleting* it, and that is not what you wanted in this example here.

The first **JOIN** joins lines on the left side, and the second joins on the right side; the user-defined join-point character then disappears.

```
CHANGE P' [AB-CD-EF]' P'?!?' ALL .ONE .ONE
JOIN P'?!' '' ALL .ONE .TWO
JOIN P'?!' '' ALL .ONE .TWO
```

By the way, if you don't feel like using the ? character, pick anything you like that's convenient and not already in your data.

Line joining and line exclusion

The **JOIN** command supports the **X** and **NX** keywords, to allow you to limit your line-range selection to only excluded (**X**), or only not-excluded lines (**NX**), if you wish. Regardless of the use of **X** or **NX** keywords, when a line is joined-to, it is considered a "change" to the that line.

- When a left-join is done, a given line is joined to a prior line; that prior line is the "joined-to" line.
- When a right-join is done, a given line is joined to a subsequent line; that subsequent line is the "joined-to" line.

Any joined-to line that was excluded at the time the join is done will be **unexcluded**.

The **JOIN** command does **not** support the **MX** and **DX** keywords at this time.

Simulating **CHANGE** operands that are not available on **JOIN**

As noted above, **JOIN** does not allow **MX** and **DX**. **JOIN** also does not presently support a column range. Suppose you wanted such features for your **JOIN** processing; how could you accomplish it?

The main way is to use other commands in a "pre-join preparatory step" and then do your **JOIN**.

To achieve the effect of **MX** or **DX**, it may be possible to use a **FIND** or **CHANGE** with **MX** or **DX** first.

Another approach is to use the **TAG** command, tagging lines where you want the **JOIN** to take place, then going back and using **JOIN** with the tag name you set. See [TAG - Alter Tag Status of a Range of Lines](#) for more information on using this command.

For example, suppose you want to do a "left join" on all lines where the string ABC appears in columns 21 to 29. But, ABC might be anywhere in those columns, and it might be preceded by any arbitrary text, so there is no easy way to do this. Here is an example of how you could do this:

First, tag all the lines that meet your criteria. Let's use a tag of :J for "join", and we will use the **SET** option of **TAG** so any prior :J tags will be cleared.

Then, we can do the join on the tagged lines. Because the prior TAG command only tagged lines having the ABC string, none of those lines would ever be of zero length, so the **JOIN** command with the arguments of **P'['='] P'='** will always 'find' every line that we just tagged with :J.

Finally, if you wish, we can clear out the :J tags since they are no longer needed.

Here are the commands you would use:

```
TAG :J SET 'ABC' 21 29 ALL
JOIN P'['='] P'=' ALL :J
RESET TAG :J
```

KEEP - Retain Specified Lines in a Tag Group

Syntax

KEEP	line-tag
	[FIRST LAST]

Operands

line-tag The specific line tag identifying the line tag blocks which are to be processed by the command. Line tags are a method of marking possibly noncontiguous lines as belonging to a related 'group'. A full discussion of line tags and how to use them is in ["Working with Line Tags"](#).

FIRST only the first line in each tag block is retained; all other lines in each tag block are deleted

LAST only the last line in each tag block is retained; all other lines in each tag block are deleted

Description

The **KEEP** command only pertains to "proper" tag-blocks, which are contiguous groups of 2 or more lines having the same line tag. When such a block is 'kept', one or more lines are deleted from the block. When multiple tag blocks having the same tag exist, each tag block is processed independently. The **KEEP** command applies its processing to every tag block in the entire edit file having the specified line tag.

A lone line having a tag which is not adjacent to another line having the same tag is not considered a proper tag-block, and will be ignored by the **KEEP** command.

KEYMAP - Display Keyboard Settings Dialog

Syntax

```
KEYMAP [ LIST ]
```

Operands

LIST Requests a **KEYMAP LIST** to be created, formatted, stored in the Windows clipboard, and then a CLIP Edit window is opened in SPFLite, to allow you to view and analyze the current KEYMAP settings for all keys.

Abbreviations and Aliases

KEYMAP can also be spelled as **KEYS**, **KEY** or **KBD**

Description

The **KEYMAP** command requests the display of the SPFLite Keymap dialog. This dialog allows you to completely customize the operation of the keyboard.

You may assign primitive keyboard functions, primary commands and line commands to any desired key or key combination via this dialog. As well, keyboard macros can be created and assigned to keys. A keyboard macro is a sequence of normal key entry operations that will be 'played back' when a key is pressed.

Full details of keyboard setup and customization can be found under ["Keyboard Customization and Keyboard Macros"](#).

Using KEYMAP LIST

In order to help you examine the contents of your entire "keymap inventory", you can issue the command **KEYMAP LIST**. When you do this,

- SPFLite gathers all known information about the mapping of every key
- It takes that information and formats it into a text file
- That file is stored into the Windows clipboard
- A CLIP Edit window is opened in SPFLite to allow you to examine the contents of your KEYMAP data

Because you are in an SPFLite-controlled edit screen when this happens, you have available to you all the editing features of SPFLite, including the ability to **FIND**, **SORT**, **EXCLUDE**, and so on. You can create a permanent file by issuing a **SAVEAS** command, or paste the data into an application outside of SPFLite. The data you see can be changed in any way you like; changing it will **not** affect SPFLite's KEYMAP system. Only the KEYMAP dialog can do that. So you can safely do anything you wish to this data.

This data will be displayed in two columns per line:

- In column 1 will appear the name of the key, such as **F1**, **Ctrl-F1**, **Shift-Ctrl-F1**.
- In column 25 will appear the currently assigned definition for the key named in column 1 of that line.
- Any keys that solely contain an assignment of [\(Null\)](#) will not appear in the list, because [\(Null\)](#) means to take no action whatsoever.
- A key that is assigned to [\(PassThru\)](#) will appear in the list, because [\(PassThru\)](#) means the key is defined to perform its normal action.

LC - Lower-Case a Range of Lines

Syntax

```
LC          [ line-control-range ]
           [ X | NX | ALL]
           [ U | NU ]
           [ MX | DX ]
```

Operands

line-control-range The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

X | NX Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be processed, **NX** requests only non-excluded lines are to be processed. If neither **X** or **NX** are specified, all lines in the range will be processed.

U | NU Specifies a subset of the line range to be processed. **U** requests only User lines are to be processed, **NU** requests only non-User lines are to be processed. If neither **U** or **NU** are specified, all lines in the range will be processed.

MX **MX** requests that all lines which are processed be excluded from the display following command processing. **MX** = Make excluded

DX **DX** requests that lines which are processed, which, if excluded, would normally be made visible, be left in their excluded status. **DX** = Don't change excluded status

Description

The lines processed will have all text converted to lower-case.

LINE - Apply Line Command

Syntax

```
LINE          line-command
             [ line-control-range ]
             [ FIRST | LAST | ALL ]
             [ X | NX ]
             [ U | NU ]
             [ TOP ]
```

Operands

line-command	Any standard SPFLite Edit line command. The command may be quoted or unquoted, and must be between 1 and 6 characters long. To use LINE to erase an existing line command on a line, line-command may be a blank enclosed in quotes.
	If the line-command operand contains a command that overlaps the remaining syntax of LINE , the line-command must be enclosed in quotes; otherwise, quotes are optional. See discussion below.
line-control-range	The range of lines which are to be processed by the line-command operand. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
	NOTE: the full range of line-control-range operands is available on the command line. However , using line command ranges themselves to specify the applicable line range is not allowed. There is simply too much possibility for confusion and incorrect handling between such line range specification, and the line commands being introduced by the LINE command itself.
FIRST LAST ALL	When the line-control-range applies to more than one line, this keyword decides which line(s) are affected. If omitted, ALL eligible lines are affected. When the line-control range applies to only a single line, such as a line-label like .ABC or a pseudo line-label like .123 , only that line is affected, regardless of which of these keywords, if any, are specified.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is

always positioned as the **top** line of the screen, regardless of its current location.

Description

The **LINE** primary command provides a means of applying a given Edit line-command to one or more target lines.

You can specify the target line(s) by the **line-control-range**, **FIRST | LAST | ALL** and **X | NX** operands. **ALL** is assumed if **FIRST** or **LAST** is omitted.

The operands of the **LINE** primary command may be specified in any order.

Note that the **line-control-range** and **X | NX** operands are both optional, but at least one of them must be specified. Otherwise, you will receive an error message, **No line reference or line range specified**.

For example, the **LINE** primary command

```
LINE R .10
```

will repeat (duplicate) the contents of line 10, just as though you had put an **R** line command on line 10 and then pressed Enter.

The **LINE** primary command cannot be repeated by using the commands **RFIND**, **RLOC**, **RLOCFIND** or **RCHANGE**. However, the **LINE** primary command may be retrieved and re-executed, like any other primary command, using the **RETRIEVE** command.

If the **line-command** operand contains a command that overlaps the syntax of the **LINE** primary command, then the **line-command** operand must be quoted. This means:

- To put an **X** line command on line 123, **LINE X .123** will not work. You must use **LINE 'X' .123** or **LINE X1 .123** instead.
- To put a line label **.ABC** on line 123, **LINE .ABC .123** will not work. You must use **LINE '.ABC' .123** instead.
- To put a line tag **:ABC** on line 123, **LINE :ABC .123** will not work. You must use **LINE ':ABC' .123** instead.

See [Working with the LINE Primary Command](#) for more information on using this feature.

Limitations of T/TT line command and LINE primary command

The **T/TT** line command can be used by the **LINE** primary command, to apply **T/TT** to one or more lines. However, when **T** is applied to more than one line, each individual **T** is applied to each line one at a time. The way that **T/TT** operates, only a single, contiguous block of highlighted data may exist as any given time. So, if you attempted to issue a command like **LINE T .11 .13 ALL**, only line 13 will be highlighted. If you try to manually highlight line 11 with a **T**, then line 12, then line 13, you will see how and why it works this way.

LOCATE - Scroll to a Specific Line

Syntax

Specific Format:

```
LOCATE { line-label | line-num }
```

Generic Format:

```
LOCATE [ { { line-label-a | line-num-a } { line-label-b | line-
num-b } } ]
[ NOT ]
{ CHANGE | COMMAND | ERROR | EXCLUDED | LABEL |
  NOTE | xNOTE | ZNOTE | U | NU
  FIND | SPECIAL | KEEP | TAG | LONG [ n ] |
  PAGE [ n ] | FILE | SIZE n | tagname }
[ NEXT | FIRST | LAST | PREV | CURRENT | ALL
[ MX ] ]
[ colorname | STD ]
[ TOP ]
```

Operands

line-label | line-num The line number or line label which you wish located and scrolled to the top of the screen when specifying a specific format **LOCATE** command.

line-label-a | line-num-a An optional range of lines to be searched when using the generic format **LOCATE** command.
line-label-b | line-num-b

NOT This will negate or reverse the generic search request to locate the next line which does **not** meet the specified criteria.

CHANGE Locate the next line flagged as modified by Change processing. i.e. a line marked by **==CHG>**.

COMMAND Locate the next line containing an unprocessed line command.

ERROR Locate the next line flagged as in error. i.e. a line marked by **==ERR>**. (Future use).
LOCATE ERROR is not currently supported. You can issue a command of **LOCATE ERROR**, but nothing will be found.

EXCLUDED Locate the next excluded line.

FILE Locate the next line marked by **=FILE>** marker.

FIND	Locate the next line which has been previously found by a FIND command, or changed by a CHANGE command. Even though there are no visible markers for found lines as there are for changed lines, SPFLite will 'remember' the location of found lines as if a 'hidden' marker existed on such lines. Just as with the ==CHG> marker, the hidden found-markers are cleared in response to a RESET command.
KEEP	Locate the next line containing a 'kept' line command.
LABEL	Locate the next line which contains a label.
LONG [n]	Locate the next line longer than the specified value (or when omitted, the current LRECL value when LRECL is > 0).
NOTE	Locate the next line having a = NOTE > marker.
xNOTE	Locate the next line having an extended xNOTE > marker, where "x" is any letter from A to Y .
ZNOTE	Locate the next line having an extended xNOTE > marker of any kind , where "x" is any letter from A to Y . When ZNOTE is specified, "plain" = NOTE > markers are not located. As a reminder, extended xNOTE > markers of type 'Z' are not permitted; that is, you cannot issue a line command of ZNOTE and no ZNOTE > line markers can be created.
SIZE n	Locate the next line whose length is equal to the specified value (0 is permitted to search for empty (null) lines).
PAGE [n]	Locate the next line having a = PAGE > marker. If the optional 'n' page number is provided, it will scroll to the requested page number.
SPECIAL	Locate the next special line (COLS, BNDS, TABS etc.)
TAG	Locate the next line containing a Tag
:tag	Locate the next occurrence of the specified Tag.
<u>NEXT</u> FIRST LAST PREV	These modify the search action from the normal default of NEXT and refer to a line's position within the file.
ALL [MX]	Specify ALL when LOCATE is used for the side-effect of changing the exclusion status of one or more lines. The ALL keyword will also report on the number of lines found. LOCATE ALL will locate lines matching the specified condition, and will then unexclude them. LOCATE ALL MX will locate lines matching the specified condition, and will then make them excluded.
 The keywords FILE and ALL cannot be used together. See discussion below.	
CURRENT CURR	May be used only with LOCATE FIND and LOCATE CHANGE to locate the most recently found or changed line processed by these commands.
colorname	Used to locate lines having the specified color, as set by a corresponding (Pen/colorname) "virtual highlighting pen" keyboard function. May be used with the NOT keyword to locate lines not having the specified color. To "have" a color means that at least one character on

the line has the specified color. It is not necessary for the entire line to be of that color.

See [Working with Virtual Highlighting Pens](#) for more information.

STD

Used to locate lines **not** having any of the colors that can be set by a [\(Pen/colorname\)](#) "virtual highlighting pen" keyboard function. That is, lines that are located are ones that only consist of the **standard** text color. When used with the **NOT** keyword, it can be used to locate lines having any text marked by a virtual highlighting pen in any of the standard colors. Because **LOCATE** locates **lines** and not any particular character string, a **line** is considered "standard" when there are no characters on it in **any** of the standard colors. Since at least one character must be present for a colorized character to exist, lines of length zero are always considered "standard" lines (because no characters of any of the other standard colors were present on the line).

See [Working with Virtual Highlighting Pens](#) for more information.

TOP

Normally, at the completion of the command, the first or only line located is highlighted (if it is on the current screen) or the screen is scrolled to the next appropriate screen line (as ISPF does) if the desired line is not on the current screen.

If **TOP** is coded, then the line is always positioned as the **top** line of the screen, regardless of its current location. Using **TOP** can be helpful if you are scrolling through a large file containing many instances of repetitive data; **TOP** will prevent the appearance of the screen from "bouncing around" as you go through the file, making the data easier to read.

Abbreviations and Aliases

LOCATE can also be spelled as **L** or **LOC**

CHANGE can also be spelled as **C**, **CHG** or **CHA**

COMMAND can also be spelled as **CMD** or **COM**

CURRENT can also be spelled as **CURR**

EXCLUDED can also be spelled as **X**, **EX**, **EXC**, or **EXCLUDE**

LABEL can also be spelled as **LAB** or **LABELS**

SPECIAL can also be spelled as **SPE**

TAGS can also be spelled as **TAG**

Description

The **LOCATE** command repositions the visible portion of the file to the location specified by the operands.

The **ALL** option allows you to use **LOCATE** to exclude or unexclude lines based on a "locate condition". When **ALL** is used on **LOCATE**, no particular line is located. In practice, the edit screen will be positioned at the last line located. For example, suppose the condition you were interested in was whether lines had labels. Then, you can issue commands like this:

LOCATE ALL LABEL

Find all occurrences of labeled lines, and as a side-effect, unexclude all of them.

LOCATE ALL LABEL MX

Find all occurrences of labeled lines, and as a side-effect, make all of them excluded. The **MX** option is allowed only when **ALL** is present.

Note: In a **Multi-Edit session**, the **=FILE>** marker lines cannot be manually excluded or tampered with. To maintain the integrity of these lines, you cannot issue the command **LOCATE ALL FILE MX**, nor (for sake of

consistency) can you say **LOCATE ALL FILE** without the **MX** either. **LOCATE ALL FILE** would only have reported the number files in the Multi-Edit session, and this information is available on the status line.

Note: **LOCATE ERROR** is reserved for future use to support ISPF-compatible functionality, but is not currently supported.

Examples

To find the next special line

LOCATE SPE

To find the next line with a label

LOC NEXT LABEL

To find the next line with a tag

LOC NEXT TAG

To find the last line with a tag of :T

LOC LAST :T

To find the next excluded line between **.START** and **.END**

LOC X .START .END

To find the first excluded line between **.E** and **.S**

L FIRST .E .S X

To find the first line that is exactly of length 25

L FIRST SIZE 25

To find the first line with text that **is** marked with the Green virtual highlighting pen color

L FIRST GREEN

To find the last line with text that **is not** marked with the Red virtual highlighting pen color

L LAST NOT RED

To find the first line with text that **is not** marked with any virtual highlighting pen color

L FIRST STD

To find the last line with text that **is** marked with any virtual highlighting pen color

L LAST NOT STD

LRECL - Specify Record Length

Syntax

LRECL	n
--------------	----------

Operands

n	The (maximum) Logical Record Length of the file, or 0 (zero)
----------	---

Description

Conventional Windows text files are simple text lines delimited by an End-Of-Line sequence, usually the control characters CR/LF.

Text records may be any length from zero up to the systems maximum string length (2 gig), though in practice, text lines are never so long. A conventional Windows text file has the following record format information in its Profile:

- **EOL CRLF** - This is the standard CR/LF (X'0D0A') line delimiter used in Windows text files.
- **RECFM U** - **RECFM** is an IBM mainframe acronym, meaning **Record Format**, and **U** stands for Undefined or Unknown, and means it is not known in advance what the maximum size of a record is until the file is read.
- **LRECL 0** - **LRECL** is an IBM mainframe acronym, meaning **Logical Record Length**. For SPFLite, **LRECL** generally means the **maximum** logical record length. Specifying **LRECL 0** does not mean the maximum is zero, but rather, that there **is no** (arbitrary) maximum length.

Files originating on a mainframe or a nonstandard system may contain a fixed-length record format. The **LRECL** parameter allows you to specify a given fixed length for a record. For **fixed** length files using **RECFM F**, the **LRECL** value is then, not the **maximum** record length, but the **only** record length.

Fixed-length records may be stored either with or without End of Line characters. The **LRECL** size refers to the data length, without regard to the presence of delimiters.

If you wish to edit a file with no End of Line delimiters at all, you would specify **EOL NONE**. File types declared to be **EOL NONE** must have a defined **LRECL** value that is not **0** (zero). You would also have to **RECFM F** to signify a fixed-length file, or **RECFM V** to signify a variable-length file that contains Record Descriptor Words (RDW's) instead of EOL delimiters.

See also [MINLEN - Set Minimum Record Length](#) and [Managing Line Lengths](#) for more information.

More details on handling special file formats can be found in [Handling Non-Windows Text Files](#).

The **LRECL** value is stored as part of the PROFILE options which are maintained individually by file type.

MAKELIST - Create FILELIST

Syntax

```
MAKELIST      list-name      [ SYM ]
                  [ REPLACE ]
                  [ APPEND ]
```

Operands

list-name	The name of the File List you wish to create or replace.
SYM	If SYM is coded, the list will be created as set of generic paths, rather than a list of specific file names.
REPLACE	If REPLACE is coded, then if list-name.FILELIST exists, it will simply be replaced.
APPEND	If APPEND is coded, the current set of files will be merged with the existing contents of list-name.FILELIST

Abbreviations and Aliases

MAKELIST can also be spelled as **ML**

REPLACE can also be spelled as **REPL** or **REP**

Description

The **MAKELIST** command can only be used on the File Manager screen. Its purpose is to allow saving the current list of files being displayed as a File List for quick access in the future. This can be helpful when you have managed to create a unique list of files say perhaps after doing some **FF** (Find in Files) commands, and want to refer to this list in the future. You can also use it to 'take a snapshot' of a directory listing.

Simply enter **MAKELIST myname** and the displayed list of files will be saved as **myname.FLIST**. If **myname.FLIST** currently exists, and you wish to replace it with a new set of contents, then also code the **REPLACE** operand.

Any File List created by **MAKELIST** is considered the same as a File List created by the **FAVORITE** command. That means your newly created File List will be created as a **Named Favorites**.

At any time in the future you can re-display this File List with a command of **RECALL myname** on any primary command line. You can also click on the File List name you created underneath the **Lists** item in the Quick Launch bar by selecting it with the mouse.

If the **APPEND** operand is coded, the current list of files will be merged with the existing contents of the named **FILELIST**.

Making a Symbolic FILELIST

When the **SYM** operand is used, it requests a directory list to be saved as a series of generic or symbolic pathnames. This is done by extracting each unique path displayed in the list. Thus, when the list is selected in the

future, it will display the files that exist **then**, rather than the files that exist **now**. If you edit a symbolic File List using the **E** line command, you will see a list of path names, one per line, with one entry per unique path that was present when you created the File List.

Reserved FILELIST names

The File List names **FOUND**, **OPEN** and **PATHS** are reserved for displaying the special lists described in [RECALL](#). This is in addition to **RECENT**, **FAV**, **FAVORITE** and **FAVOURITE** which were already reserved.

Reserved File List names cannot be used for user-defined named favorites for the [FAVORITE](#) and [MAKELIST](#) commands. Even though you can issue a command like **RECALL OPEN**, you cannot say **FAVORITE OPEN**, because that would imply you were creating (or adding a file name to) a **Named Favorites** File List called **OPEN**, which SPFLite has reserved for its internal list of currently-open files.

MARK - Turn Mark ON or OFF

Syntax

MARK	[<u>ON</u> OFF]
-------------	----------------------------

Operands

ON | OFF The desired MARK status

Description

The [MARK line command](#) allows you to display faint vertical lines on the edit screen to assist in the left / right positioning of columnar data.

However, having to clear and/or re-establish Mark positions every time you wanted to work with (or without) them would be burdensome.

The MARK [primary command](#) enables or disables mark support, without altering the currently-saved column settings previously established by the [MARK line command](#).

The **ON/OFF** value of the [MARK primary command](#) is stored as part of the PROFILE options which are maintained individually by file type.

MEDIT - Add a File To a Multi-Edit Session

Syntax

MEDIT	[filename]
--------------	---------------------

Operands

filename If **filename** is not specified, you will be presented with a Windows open dialog, and asked to specify the file to edit.

Description

If the current tab when the **MEDIT** command is issued is already a multi-edit session, a new **=FILE>** separator line will be added after the last data line in the last file, and then the data lines from the newly added file will appear after that.

If the current tab is **not** currently a multi-edit session, it will be converted to one and the file specified will be added to it as described above.

See [Working with Multi-Edit Sessions](#) for more information.

MINLEN - Set Minimum Record Length

Syntax

MINLEN	n
--------	---

Operands

n The Minimum logical record Length of the file, or **0** (zero)

Description

MINLEN defines the minimum length for lines in a file. When **MINLEN** is greater than zero, whenever the file is edited and lines are inserted, modified or copied from the clipboard for from another file, the minimum line length will be enforced, by blank-padding any lines that are shorter than **MINLEN** characters.

When the **MINLEN** option is set, SPFLite does the following:

- If the **MINLEN** value is greater than zero, the file currently being edited is scanned, any any lines shorter than the newly-specified **MINLEN** are padded with blanks to the minimum length.
- Any lines changed manually by typing into them, lines changed from primary or line commands, and lines added from **COPY** and **PASTE** commands, will have their minimum line length enforced by the **MINLEN** value in effect at the time.

One reason to use a **MINLEN** value greater than zero is to avoid the existence of zero-length lines, which can create certain issues with **FIND** and **CHANGE** pictures.

The installation default for **MINLEN** is **0**.

See also [LRECL - Specify Record Length](#) and [Managing Line Lengths](#) for more information.

More details on handling special file formats can be found in [Handling Non-Windows Text Files](#).

The **MINLEN** value is stored as part of the PROFILE options which are maintained individually by file type.

NDELETE - Delete Lines Where String is Not Found

Syntax

```
NDELETE      string [ PREFIX | SUFFIX | WORD | CHAR ]
                [ start-column [ end-column ] ]
                [ line-control-range ]
                [ color-selection-criteria ]
                [ X | NX ]
                [ U | NU ]
                [ ALL | FIRST | LAST | NEXT | PREV ]
                [ TOP ]
```

Operands

string

For **NDELETE**, the string option is required; unlike the **DELETE** command, all **NDELETE** operations are string-based. Lines not containing string are deleted within the selected line range. If neither **ALL**, **FIRST**, **LAST**, **PREV** or **NEXT** is specified, **NEXT** is assumed. The string may be unquoted, or simply quoted, or may be any of the standard string types **C**, **T**, **X**, **P** or **R**.

CHARS | **WORD**

|

PREFIX |
SUFFIX

These options describe the “string context” in the same way that a **FIND** or **CHANGE** command does.

If omitted for a string-based delete, the string will be searched for in **WORD** mode if **C W** or **T W** appears in the middle of the status line, and in **CHARS** mode if **C** or **W** appears in the middle of the status line. **WORD** mode or **CHARS** mode can be set by **FIND WORD** or **FIND CHARS**, respectively.

start-column

Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.

end-column

Right column of a range (with start-column) within which the search-string value must be found.

line-control-range

The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#). Refer to that section of the documentation for details.

color-selection-criteria

A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in ["Color Selection Criteria Specification"](#). Refer to that section of the documentation for details.

ALL

All lines in the line range not having the search string are deleted. Unlike the **DELETE** command, **ALL** is not assumed for **NDELETE** if **ALL**, **FIRST**, **LAST**, **NEXT** and **PREV** are omitted.

X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be deleted, NX requests only non-excluded lines are to be deleted. If neither X or NX are specified, all lines in the range will be eligible to be deleted.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
FIRST NEXT	Starts at the top of the specified range and searches ahead to find the first occurrence in the specified line-control-range and delete that line. NEXT is assumed if ALL , FIRST , LAST , NEXT or PREV are omitted.
LAST PREV	Starts at the bottom of the specified range and searches backward to find the last occurrence in the specified line-control-range and delete that line.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

NDELETE can also be spelled as **NDEL**
PREFIX can also be spelled as **PRE** or **PFX**
SUFFIX can also be spelled as **SUF** or **SFX**
WORDS can also be spelled as **WORD**
CHARS can also be spelled as **CHAR**

Description

NDELETE removes one or more lines from the edit file where a given string is **not** found.

Specifying the Lines to NDELETE

Two methods of specifying the data are possible.

- The first is the classic ISPF method of specifying line numbers as operands to the **NDELETE** command. You have the full range of options allowed by SPFLite's extended line-control-range operands.
- Or, the lines may be marked by **C/CC** line commands.

Depending on whether the **X** or **NX** operands are specified, the data deleted will be the entire contents of the specified line range if these operands are omitted, or the specified subset (**X** or **NX**) if the operands are specified.

The options **ALL**, **FIRST**, **LAST**, **PREV** and **NEXT** are all available.

Examples of the NDELETE command

To delete all excluded lines not containing ABC:

```
NDELETE ABC ALL X
```

To delete all non-excluded lines not containing ABC:

```
NDELETE ABC ALL NX
```

To delete a range of lines not containing ABC using line numbers:

```
NDELETE ABC ALL 20 30
```

To delete a range of lines not containing ABC using line labels:

```
NDELETE ABC ALL .HERE .THERE
```

To delete only excluded lines not containing ABC within a range of lines using line numbers:

```
NDELETE ABC ALL X 25 35
```

To delete lines not containing ABC tagged with an :AB line tag:

```
NDELETE ABC ALL :AB
```

To NDELETE all unexcluded lines not containing the text "abc" as a prefix:

```
NDELETE ALL T'abc' PREFIX NX
```

Deleting all-blank lines with NDELETE

It is not possible to use **DELETE** to delete all blank lines. The reason is that a "blank" might have a variable number of space characters, or in a zero-length line there are no space characters at all. So, there is no specific string you could "search" for that would always be there.

The easiest way to delete blank lines is to use **NDELETE** instead of **DELETE**. Instead of trying to find "blank" lines, you find and then delete all lines which are **not** non-blank lines.

People sometimes have issues with this, since it's like a "double negative", and not intuitive. Once you observe it in action, you'll see that it makes sense.

You can find **non-blank** lines by finding lines that contain `P'^'` or `P'-'`, which match non-blank characters. So, to delete all **blank** lines, you delete all lines which **don't** contain any **non-blanks**, like this:

```
NDELETE P'^' ALL
```

Be sure to specify **NDELETE** and not **DELETE**. If you say `DELETE P'^' ALL`, you will delete every line in the file that is not blank. In other words, you will have just deleted all of your data!

If you don't care for the "double negative" appearance of this command, you can define a **SET** variable that's easier to remember. Let's say we call this variable **DBLANK**. You define it like this (the outer quotes are required):

```
SET DBLANK = "NDELETE P'^'"
```

Then when you want to delete all blank lines, you would do it like this:

```
=DBLANK ALL
```

You can create a command alias so that the leading = sign is not needed. To do this, define the **DBLANK** command like this:

```
SET ALIAS.DBLANK = "NDELETE P'^'"
```

Then when you want to exclude all blank lines, you would do it like this:

```
DBLINK ALL
```

NEXCLUDE - Exclude Where String is Not Found

Syntax

```
NEXCLUDE      search-string
                  [ start-column [ end-column ] ]
                  [ FIRST | LAST | NEXT | PREV | ALL ]
                  [ PREFIX | SUFFIX | WORD | CHAR ]
                  [ line-control-range ]
                  [ U | NU ]
                  [ color-selection-criteria ]
                  [ TOP ]
```

Operands

search-string	The search string that identifies the lines whose exclusion status will remain unchanged. Lines where this string is not found will be excluded
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first non -occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last non -occurrence of search-string.
NEXT	Starts at the first position after the current cursor location and searches ahead to find the next non -occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous non -occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all non -occurrences of search-string.
PREFIX	Looks for search-string at the beginning of a word.
WORD	Looks for search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
CHAR	Locates search-string regardless of what precedes or follows it.
SUFFIX	Looks for search-string at the end of a word.

line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

NEXCLUDE can also be spelled as **NX**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the **NEXCLUDE** command to look for a search string, and exclude (conceal) the lines that **do not** contain the string from the display. Note that the normal selection options of **X** and **NX** are not allowed, since **NEXCLUDE** operates **only** on non-excluded lines. See [Working with Excluded Lines](#) for more information.

To exclude the next non-excluded line that **does not** contain the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

NEXCLUDE ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming that **CASE T** is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To exclude the next line that **does not** contains the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

NEXCLUDE C"ELSE"

and press Enter.

This type of exclusion is called a character string exclusion (note the **C** that precedes the search string) because it searches for the letters ELSE only if the letters are found in uppercase. However, since no other qualifications were specified, the exclusion occurs no matter where the letters are found on a non-excluded line, as outlined in the previous list.

Excluding all-blank lines with **NEXCLUDE**

It is not possible to use **EXCLUDE** to exclude all blank lines. The reason is that a "blank" might have a variable number of space characters, or in a zero-length line there are no space characters at all. So, there is no specific string you could "search" for that would always be there. SPFLite's string search engine cannot find strings of zero length, because there is literally nothing to find.

The easiest way to exclude blank lines is to use **NEXCLUDE** instead of **EXCLUDE**. Instead of trying to exclude "blank" lines, you exclude all lines which are **not** non-blank lines.

People sometimes have issues with this, since it's like a "double negative", and not intuitive. Once you observe it in action, you'll see that it makes sense.

You can find **non-blank** lines by finding lines that contain `P'^'` or `P'-'`, which match non-blank characters. So, to exclude all **blank** lines, you exclude all lines which **don't** contain any **non-blanks**, like this:

```
NEXCLUDE P'^' ALL
```

Be sure to specify **NEXCLUDE** (or **NX**) and not **EXCLUDE**. If you say **EXCLUDE P'^' ALL**, you will exclude every line in the file that is not blank.

If you don't care for the "double negative" appearance of this command, you can define a **SET** variable that's easier to remember. Let's say we call this variable **XBLANK**. You define it like this (the outer quotes are required):

```
SET XBLANK = "NX P'^'"
```

Then when you want to exclude all blank lines, you would do it like this:

```
=XBLANK ALL
```

You can create a command alias so that the leading = sign is not needed. To do this, define the XBLANK command like this:

```
SET ALIAS.XBLANK = "NX P'^'"
```

Then when you want to exclude all blank lines, you would do it like this:

```
XBLANK ALL
```

NFIND - Find Where String is Not Found

Syntax

```
NFIND           search-string
                  [ start-column [ end-column ] ]
                  [ FIRST | LAST | NEXT | PREV | ALL ]
                  [ PREFIX | SUFFIX | WORD | CHAR ]
                  [ line-control-range ]
                  [ X | NX ]
                  [ U | NU ]
                  [ MX | DX ]
                  [ TOP ]
```

Operands

search-string	The search string that identifies the lines to be skipped over, the search will be satisfied by the line which does not contain this string.
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first line which does not contain the search-string.
LAST	Starts at the bottom of the data and searches backward to find the first line which does not contain the search-string.
<u>NEXT</u>	Starts at the first position after the current cursor location and searches ahead to find the first line which does not contain the search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the first line which does not contain the search-string.
ALL	Starts at the top of the data and searches ahead to find all lines which do not contain the search-string.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.

line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be examined. U requests only User lines are to be examined, NU requests only non-User lines are to be examined. If neither U or NU are specified, all lines in the range will be examined.
MX	MX requests that all lines which do contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which do contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't alter Excluded status.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

NFIND can also be spelled as **NF**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the **NFIND** command to locate line(s) within the file which do **not** contain a specified string.

To find the next line which does not contain the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

NFIND ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming that **CASE T** is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word).
- In either an excluded or a non excluded line.
- Anywhere within the current boundaries.

To find the next line which does **not** contain the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

NFIND C'ELSE'

Press Enter.

This type of search is called a character string search (note the **C** that precedes the search string) because it finds the next occurrence of the letters ELSE only if the letters are in uppercase. However, since no other qualifications were specified, the letters can be found anywhere in the file, as outlined in the preceding list.

For more information, including other types of search strings, see [Finding and Changing Data](#) and [Specifying a Picture or Format String](#).

NOTE: When the **NFIND** search operand is a Regular Expression string, certain search operands are restricted. See [Specifying A Picture String](#) for more information.

Finding all-blank lines with **NFIND**

It is not possible to use **FIND** to find all blank lines. The reason is that a "blank" might have a variable number of space characters, or in a zero-length line there are no space characters at all. So, there is no specific string you could "search" for that would always be there. SPFLite's string search engine cannot find strings of zero length, because there is literally nothing to find.

The easiest way to find blank lines is to use **NFIND** instead of **FIND**. Instead of trying to find "blank" lines, you find all lines which are **not** non-blank lines.

People sometimes have issues with this, since it's like a "double negative", and not intuitive. Once you observe it in action, you'll see that it makes sense.

You can find **non-blank** lines by finding lines that contain `P'^'` or `P'-'`, which match non-blank characters. So, to find all **blank** lines, you find all lines which **don't** contain any **non-blanks**, like this:

```
NFIND P'^' ALL
```

Be sure to specify **NFIND** and not **FIND**. If you say `FIND P'^' ALL`, you will find every line in the file that is not blank.

If you don't care for the "double negative" appearance of this command, you can define a **SET** variable that's easier to remember. Let's say we call this variable **FBLANK**. You define it like this (the outer quotes are required):

```
SET FBLANK = "NFIND P'^'"
```

Then when you want to find all blank lines, you would do it like this:

```
=FBLANK ALL
```

You can create a command alias so that the leading = sign is not needed. To do this, define the **FBLANK** command like this:

```
SET ALIAS.FBLANK = "NFIND P'^'"
```

Then when you want to exclude all blank lines, you would do it like this:

```
FBLANK ALL
```

NFLIP - Negative Reverse Exclusion Status of Lines

Syntax

```
NFLIP           search-string
                  [ start-column [ end-column ] ]
                  [ FIRST | LAST | NEXT | PREV | ALL ]
                  [ PREFIX | SUFFIX | WORD | CHAR ]
                  [ line-control-range ]
                  [ U | NU ]
                  [ color-selection-criteria ]
                  [ TOP ]
```

Operands

search-string	The search string that identifies the lines to be exempted from being flipped
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of search-string.
NEXT	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
CHAR	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line

[Control Range Specification](#)". Refer to that section of the documentation for details.

U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

All commands that affect the exclusion status of lines are "unified" with a common design patterned after the **EXCLUDE** command. That makes all of the commands very powerful. The **NFLIP** command formerly applied to all lines by default, but now must use **ALL** if you want to affect all lines.

You can use the **NFLIP** command to find those lines which **do not** have a search string, and invert the visibility state of those lines. i.e. exclude the line if currently visible, or show the line if currently excluded. Note that the normal selection options of **X** and **NX** are not allowed, since **NFLIP** must process both types of lines. You may also wish to review ["Working with Excluded Lines"](#) for more information.

To flip the status of the next line that **does not** contain the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

NFLIP ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming that **CASE T** is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To flip the status of the next line that **does not** contain the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

NFLIP C"ELSE"

and press Enter.

This type of search is called a character string search (note the **C** that precedes the search string) because it flips the next line that **does not** contain the letters ELSE only if the letters are found in uppercase. However, since no other qualifications were specified, the exclusion occurs no matter where the letters are found on a non-excluded line, as outlined in the previous list.

NOTIFY - Set Temporary File Notification Level

Syntax

NOTIFY	[ALL NONE EDIT RESET]
---------------	--------------------------------------

Operands

ALL | EDIT | Sets the desired temporary notification level

NONE

RESET Sets the notification level back to the permanent level defined in the General tab of the Global Options dialog

Description

SPFLite can be directed to inform you when a file you are working on in an Edit or Browse session has been modified by some process outside of SPFLite itself. You have control over what conditions under which you will receive a notification.

The **NOTIFY** command is used to set the temporary notification level for files opened in SPFLite that are modified by an external process. The setting you choose on **NOTIFY** only lasts until the next **NOTIFY** command, or until SPFLite is terminated.

The permanent file notification level is defined in the General tab of the Global Options dialog, and is either **ALL**, **NONE** or **EDIT**.

If none of the options **ALL**, **NONE**, **EDIT** or **RESET** are present, **NOTIFY** will report what the current notification level is.

Note: There is no relationship between the **NOTIFY** command in SPFLite and the **NOTIFY** feature in TSO/ISPF, which is used to notify TSO users that a submitted job has completed.

See [Handling External File Changes](#) for more information.

NREVERT - Negative Revert of User Line to Ordinary Line

Syntax

```
NREVERT           string
                  [ CHAR | WORD | PREFIX | SUFFIX ]
                  [ start-column [ end-column ] ]
                  [ line-control-range ]
                  [ color-selection-criteria ]
                  [ MX | DX ]
                  [ X | NX ]
                  [ ALL ]
                  [ TOP ]
```

Operands

string	The string operand is required. It provides the string which, if not found, will cause the line to be un-marked.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-Word characters.
PREFIX	Locates search-string at the beginning of a word.
SUFFIX	Locates search-string at the end of a word
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be eligible to be processed.

MX	MX requests that all lines which DO contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

NREVERT can also be spelled as **NV**
PREFIX can also be spelled as **PRE** or **PFX**
SUFFIX can also be spelled as **SUF** or **SFX**
WORDS can also be spelled as **WORD**
CHARS can also be spelled as **CHAR**

Description

NREVERT will remove ("revert") the User Line status from all lines which **do not** meet the specified criteria. This is an alternative method to using the **V** / **VV** line commands to remove the User Line status of selected lines.

When a U line reverts to an ordinary V line, the **|** vertical bar that marks the "gap column" will disappear.

Example uses of the NREVERT command

To revert all User Lines which **do not** contain DEF back to "ordinary" V lines:

```
NREVERT ALL "DEF"
```

To revert all User Lines which **do not** contain the word TEST between columns 5 and 10, and which are highlighted in GREEN, back to "ordinary" V lines:

```
NREVERT "TEST" WORD 5 10 GREEN ALL
```

For more information on User lines see "[Working with User lines](#)"

NSHOW - Show Lines Where String is Not Found

Syntax

```
NSHOW           search-string
                  [ start-column [ end-column ] ]
                  [ FIRST | LAST | NEXT | PREV | ALL ]
                  [ PREFIX | SUFFIX | WORD | CHAR ]
                  [ line-control-range ]
                  [ U | NU ]
                  [ color-selection-criteria ]
                  [ TOP ]
```

Operands

search-string	The search string that identifies the lines whose exclusion status will remain unchanged. Lines where this string is not found will be unexcluded
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first non -occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last non -occurrence of search-string.
NEXT	Starts at the first position after the current cursor location and searches ahead to find the next non -occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous non -occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all non -occurrences of search-string.
PREFIX	Looks for search-string at the beginning of a word.
WORD	Looks for search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
CHAR	Locates search-string regardless of what precedes or follows it.
SUFFIX	Looks for search-string at the end of a word.
line-control-	The range of lines which are to be processed by the command. Line control ranges

range	provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**
SUFFIX can also be spelled as **SUF** or **SFX**
WORDS can also be spelled as **WORD**
CHARS can also be spelled as **CHAR**

Description

You can use the **NSHOW** command to look for a search string, and unexclude the lines that **do not** contain the string from the display. Note that the normal selection options of **X** and **NX** are not allowed, since **NSHOW** operates **only** on excluded lines. See [Working with Excluded Lines](#) for more information.

To un-exclude the next excluded line that **does not** contain the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

NSHOW ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase (assuming that **CASE T** is in effect)
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To unexclude the next line that **does not** contains the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

NSHOW C"ELSE"

and press Enter.

This type of exclusion is called a character string exclusion (note the **C** that precedes the search string) because it searches for the letters **ELSE** only if the letters are found in uppercase. However, since no other qualifications were specified, the exclusion occurs no matter where the letters are found on a non-excluded line, as outlined in

the previous list.

NULINE - Negative Mark of User lines

Syntax

```

NULINE          string
                  [ CHAR | WORD | PREFIX | SUFFIX ]
                  [ start-column [ end-column ] ]
                  [ line-control-range ]
                  [ color-selection-criteria ]
                  [ MX | DX ]
                  [ X | NX ]
                  [ ALL ]
                  [ TOP ]

```

Operands

string	The string operand is required. It provides the string which, if not found, will cause the line to be marked.
CHAR	Locates search-string regardless of what precedes or follows it.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-Word characters.
PREFIX	Locates search-string at the beginning of a word.
SUFFIX	Locates search-string at the end of a word
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be eligible to be processed.

MX	MX requests that all lines which DO contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

NULINE can also be spelled as **NU**
PREFIX can also be spelled as **PRE** or **PFX**
SUFFIX can also be spelled as **SUF** or **SFX**
WORDS can also be spelled as **WORD**
CHARS can also be spelled as **CHAR**

Description

NULINE will add the User line status to all lines which **do not** meet the specified criteria. This is an alternative method to using the **U** / **UU** line commands to mark lines as User Lines.

When an "ordinary" V line becomes a U line , a | vertical bar will appear in the "gap column".

Example uses of the NULINE command

To mark all lines containing which do **not** have **ABC** in column 12 as User Lines:

```
NULINE ALL "ABC" 12
```

To mark all lines which do not contain the word **FRED** highlighted in **RED** as User Lines:

```
NULINE "FRED" WORD RED ALL
```

For more information on User lines see "[Working with User lines](#)"

NUMBER - Verify and generate sequence numbers

Syntax

```
NUMBER
```

Operands

NUMBER has no operands

Abbreviations and Aliases

NUMBER can also be spelled as **NUMB** or **NUM**

Description

The **NUMBER** command can be used to verify and (as needed) generate or correct sequence numbers if these are missing, invalid or out of sequence. If a file already contains a complete set of valid sequence numbers, NUMBER will not modify it. The processing performed by NUMBER attempts to modify as little of your file as possible to ensure that each line has a valid sequence number in ascending order.

All numbering commands replace data in existing column positions, with no attempt to save or shift-over your data to make room for these numbers. It is an "overlay operation", not an "insert operation". So, for instance, if you use COBOL numbering, columns 1-6 of every line of your file will be replaced by sequence numbers, **regardless of any data that may already be present there**. It is your responsibility to alter the contents of your file, if necessary, if that is not the outcome you wanted.

See [Working with Sequence Numbering](#) for more information.

NUMTYPE - Enable Numbering and Define Numbering Type

Syntax

```
NUMTYPE      [ NONE |  
             {  
                 start-column end-column  [ TRUNC ] [ REAL ] |  
                 COB | STD | IBM |  
             }  
         ]
```

Operands

NONE	Used to nullify NUMTYPE. When NUMTYPE is set to NONE, all numbering commands such as NUMBER, RENumber and UNNUMBER are ineffective. If NONE is specified, the remaining operands are not allowed.
start-column	A required nonzero start-column number where sequence numbers are stored. A sequence field may be from 4 to 8 digits wide.
end-column	A required nonzero end-column number where sequence numbers are stored.
TRUNC	Requests the data line be truncated immediately following the sequence number field.
REAL	Requests that the sequence field be set equal to the normal SPFLite line number, rather than some other incrementing number value..

Description

The **NUMTYPE** command is normally specified using the start-column - end-column syntax. As an aid, NUMTYPE will accept the aliases listed below as shortcuts for common values.

NUMTYPE COB	Equivalent to NUMTYPE 1 6
NUMTYPE STD	Equivalent to NUMTYPE 73 80
NUMTYPE IBM	Equivalent to NUMTYPE 73 80 TRUNC

Once a valid NUMTYPE specification is entered (other than NONE), it is saved in the normal File Profile. The other numbering commands (NUMBER, RENumber and UNNUMBER) will then become active and available and **will use** the specified NUMTYPE values..

All numbering commands replace data in existing column positions, with no attempt to save or shift-over your data to make room for these numbers. It is an "overlay operation", not an "insert operation". So, for instance, if you use COBOL numbering, columns 1-6 of every line of your file will be replaced by sequence numbers, **regardless of any data that may already be present there**. It is your responsibility to alter the contents of your file, if necessary, if that is not the outcome you wanted.

Right-Hand Sequence Numbers

Although it is hard to picture a normal use for this, the NUMTYPE and Numbering commands support the

specification of a sequence number field **in relation to the right-hand end of the line**. If used with a file RECFM which allows variable length lines, this could result in sequence numbers being created in a 'ragged' right margin effect. If you use this option, you had better be sure you know what you are asking for.

To request Right hand alignment, simply use negative column numbers, where the last position in a line is **-1**, the 2nd last is **-2**, and so on. For example, an 8 column sequence field at the right end would be specified as:

NUMTYPE -8 -1

Thus, if you had a RECFM=F, LRECL=80 file, the following two NUMTYPE requests would be identical.

NUMTYPE 73 80

NUMTYPE -8 -1

See [Working with Sequence Numbering](#) for more information.

OPEN - Open New SPFLite Instance and Edit File

Syntax

```
OPEN           [ filename ]
OPENB
OPENV
```

Operands

filename The name of a file to be edited. If a simple filename is entered, it will be looked for in the same directory as the file currently being edited. Of course, you may type any fully qualified name to edit a file in some other location.

Description

There are three variations of **OPEN** -> **OPEN**, **OPENB**, and **OPENV**. The only difference is the mode the file will be opened in: **EDIT (OPEN)**, **BROWSE (OPENB)** or **View (OPENV)** .

Examples

OPEN	Will open a standard Open File dialog for you to choose a file to edit. Once selected, a new SPFLite session.
OPENB SOMETEXT.TXT	Will open SOMETEXT.TXT in a new SPFLite session in Browse mode. The file will be opened in the same directory as the current file being edited.
OPEN C:\AUTOEXEC.BAT	Will open the specific file in a new SPFLite Edit mode session.
OPENV C:\SYS.INI	Will Open the SYS.INI file in a new View mode session.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the **OPEN** command, the default directory used for searching for the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the Path for **that** active file is used as the default for the command.
- If there is NO active file [when the tab header displays (Empty)], the current displayed directory of File Manager will be used.

OPTIONS - Set Global Editor Options

Syntax

OPTIONS	[<u>GENERAL</u> FILEMGR SUBMIT SCREEN SCHEMES KEYBOARD STATUS HILITES]
----------------	---

Operands

GENERAL	Open the Global Options dialog and display the General options tab
FILEMGR	Open the Global Options dialog and display the File Manager options tab
SUBMIT	Open the Global Options dialog and display the Submit options tab
SCREEN	Open the Global Options dialog and display the Screen options tab
SCHEMES	Open the Global Options Dialog and display the Schemes tab
KEYBOARD	Open the Global Options dialog and display the Keyboard options tab
STATUS	Open the Global Options dialog and display the Status Bar options tab
HILITES	Open the Global Options dialog and display the HiLites options tab

Abbreviations and Aliases

OPTIONS can also be spelled as **OPTION**, **OPT**, **EDITSET** or **EDSET**

FILEMGR can also be spelled as **FM**

KEYBOARD can also be spelled as **KB**

Description

The **OPTIONS** command will display the Global Options dialog, to allow you to modify SPFLite general settings. See ["Customizing SPFLite"](#) for details.

You can specify which tab in the Global Options dialog is opened, by specifying one of the keywords listed above. If you omit the keyword, **OPTIONS** alone acts like **OPTIONS GENERAL**, to be compatible with the behavior of prior versions of SPFLite.

You can abbreviate the keyword to any substring that uniquely identifies the tab. For **GENERAL**, **FILEMGR**, **MOUSE** and **KEYBOARD**, you can abbreviate down to one letter, like **G**, **F**, **M** and **K**. The letter **S** alone is treated like **SCREEN**, so if you want the **SUBMIT** tab, you would have to use **SUB** or at least **SU**.

You can also enter the General Options dialog using the primary command **EDITSET** or **EDSET**, which are the commands used in IBM ISPF to perform a comparable function. **EDITSET** and **EDSET** are identical in syntax and meaning to **OPTIONS**.

ORDER - Reorder File Line Numbers

Syntax

```
ORDER
```

Operands

None

Description

When the Fast Renumbering general option is in effect, the line numbers appearing in the line-command area of the edit screen may not be in contiguous order from 1 by 1 after lines are inserted and deleted. If you wish these numbers to be in contiguous order, use the **ORDER** command.

The thing that makes Fast Renumbering "fast" is that it enables SPFLite to avoid always having to renumber the entire file after changes are made. This allows the editor to renumber smaller sections of the file, and then only when it's required to maintain the line numbers in ascending order. You may see noticeable performance improvements using Fast Renumbering when dealing with very large files, such as those on the order of 100,000 lines or more.

However, the Fast Renumbering process can leave gaps in the line numbering. These numbering gaps are harmless and are essentially just cosmetic, but you may find it easier to deal with a file that does not have gaps in its line numbering, such as when you are concerned about the exact number of lines between any two points in the file.

ORDER will reorder the SPFLite line numbers from 1 by 1 throughout the entire file.

Unless you change the Fast Renumbering general option, the line numbers appearing in the line-command area may become non-contiguous again after further lines are inserted and deleted. In such cases, the effect of **ORDER** would be temporary, and would have to be issued again as needed. Because of the performance gains when dealing with large files, this situation may be an acceptable trade-off for some users.

Fast Renumbering is "in effect" or not, depending on the line-count "cutoff value" you specify in the General Options tab of the SPFLite Global Options GUI. A value of 0 forces Fast Renumbering mode for all file sizes, while a value of 999999 suppresses Fast Renumbering for all file sizes.

If you set this value to 999999, the SPFLite line numbers will always appear in order from 1 by 1, and you will not need to use the **ORDER** command. If the files you edit are usually not large, this will be the best approach.

See the [Fast Renumbering](#) General Option for more information on this feature.

See [Working with Sequence Numbering](#) for more information on sequence numbering commands.

PAGE - Set PAGE option

Syntax

PAGE	[ON OFF SCROLL]
	[[+ -] <i>offset</i>]

Operands

ON OFF SCROLL	The desired PAGE mode status
[+ -] <i>offset</i>	An optional page number adjustment factor. This affects the page # displayed in the Status Bar as well as the handling of the page number operand of LOCATE PAGE nnn. If only the offset operand is entered, ON will be assumed. The offset may not be specified if mode SCROLL is specified.

Description

The SPFLite **PAGE** setting is associated with an individual Profile, and actually is only useful when the Profile is set to **EOL=AUTO** or **EOL=AUTONL**. These EOL settings provide an automated cleanup function when loading SYSOUT files with mixed end-of-line delimiter usage. See [EOL](#) for a full description of what these EOL options support.

When **PAGE OFF** is specified, or **PAGE ON** when **EOL AUTO** is **not** also active, no special Page support will occur.

When **PAGE ON** is set, text data is displayed for only one print page at a time on the screen. i.e. if a 'page' only had a few lines, they would be displayed and the remaining lines of the screen would be left blank, just as a printed page would be left blank.

This page orientation is maintained so long as scrolling is done via **UP PAGE** and **DOWN PAGE** commands. If you use mouse-wheel or arrow key scrolling, the PAGE mode display is suspended until the next normal **UP PAGE** or **DOWN PAGE** command is issued.

If the *offset* value is specified, (as in PAGE ON +2), then SPFLite will adjust it's calculated page number by this value. A similar adjustment will be made by the LOCATE PAGE nnn command. This can aid in synchronizing the SPFLite calculated page number with the printed page numbers in the report being browsed.

When **PAGE SCROLL** is specified it acts in most respects like **PAGE ON** with the exception of page data whose number of lines exceed the number of available display lines on the screen. When this occurs, a DOWN PAGE command will **not** move to the next top-of-page marker, it will act as if DOWN FULL had been requested. This can help prevent a DOWN PAGE command from skipping large quantities of lines when the creator of the print report does not consistently honor the number of lines per page. e.g. Hercules JES2 spool output does not 'page break' the system messages log.

Note for Hercules users

Hercules users are familiar with a utility called **HercPrt**, which (among other things) has the ability to take a SYSOUT file and format it into a PDF file that looks remarkably like a computer printout on "green bar" paper - complete with sprocket holes and perforation! This is cute and very clever, but it also uses a large amount of disk space. By using alternating background colors (along with EOLAUTO and PAGE ON mode), it is possible to

very closely simulate the effect of a HercPrt-formatted file without the PDF disk-space overhead. You will still be editing or browsing ordinary text files in native mode, but the display will have the look and feel of paging through an actual hard copy printout.

If you wish to match the same colors generated by the HercPrt utility, make the main background color white, and the alternative background color a light green.

PASTE - Paste Data from the Clipboard

Syntax

```
PASTE [ name ]
[ { BEFORE | AFTER } line-label ]
[ ERASE ]
```

Operands

name	The name of a Named Private Clipboard. If omitted, the standard Windows clipboard is used.
{ BEFORE AFTER } line-label	If specified, the clipboard is copied before, or after, a defined line label. If a before/after line label is not specified, the clipboard is copied into the current edit file in one of the following ways: <ul style="list-style-type: none"> • before a B line command • after an A line command • repeatedly before the lines in a BB block • repeatedly after the lines in an AA block • into an O/OO block, overlaying the existing lines with lines from the clipboard • into an OR/ORR block, overlay-replacing the existing lines with lines from the clipboard • into an H/HH block, replacing the existing lines in that H/HH block
ERASE	If ERASE is specified, the clipboard will be erased after the PASTE operation is completed.

Description

PASTE copies data from the Windows clipboard to the current edit session.

To Paste data into an empty file:

On the Command line, type:

```
PASTE
```

Press Enter. The data is copied.

To Paste data into Edit data that is not empty:

Use the **A**, **B**, **AA**, **BB**, or **H/HH** line commands to indicate where the copied data is to be inserted. However, a number indicating that the **A** or **B** command should be repeated cannot follow the line command.

On the Command line, type:

```
PASTE
```

The **O**, **OR**, or **H** command can have a count to indicate the number of lines to be overlaid.

If the edit session is not empty, and you do not specify a destination with the **A**, **B**, **AA**, **BB**, **H/HH**, **O/OO** or **OR/ORR** line command, a error message appears in the upper-right corner of the panel, and the clipboard data is not copied.

See also information on CLIP mode, and Named Private Clipboards, in [Windows Clipboard, Cut and Paste](#).

Mapping the CUT and PASTE primary commands

Because the **CUT** and **PASTE** primary commands perform a **line-oriented** copying and pasting of data, similar to how **Ctrl-C** and **Ctrl-V** perform a **text-oriented** copying and pasting of data, many users will want to create key mappings for the **CUT** and **PASTE** commands.

In order to be of the most benefit, you may wish to define these mappings so that the current cursor location is saved and restored, so that its location doesn't "jump around" as SPFLite processes these commands.

Here are some suggested key mappings using **Alt-C** and **Alt-V** you may wish to try:

Copying lines of text to the clipboard: Use C/CC or M/MM to define the line range first:

Map **Alt-C** to: **(SaveCursor) (Home) [CUT] (Enter) (RestoreCursor)**

Pasting lines of text from the keyboard: The key mapping inserts an Aline command for you on the current line:

Map **Alt-V** to: **(SaveCursor) (LineNo) [A] (Home) [Paste] (Enter) (RestoreCursor)**

PREPEND - Insert text as start of line(s)

Syntax

```
PREPEND      string
                  [ start-column [ end-column ] ]
                  [ line-control-range ]
                  [ color-selection-criteria ]
                  [ X | NX ]
                  [ U | NU ]
                  [ ALL ]
                  [ TOP ]
```

Operands

string	The string option is required, and can only be a simple string. i.e. No Picture strings or Regular expression strings . This string will be prepended to the start of every line specified by the other PREPEND operands.
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be eligible to be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be examined, NU requests only non-User lines are to be examined. If neither U or NU are specified, all lines in the range will be examined.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Description

PREPEND adds the specified string to the beginning of all lines specified by the line-control-range, **X/NX** and **ALL** operands. The string you **PREPEND** will appear starting in column 1 of every effected line.

Example uses of the PREPEND command

To prepend the string "(.)" to all excluded lines.

```
PREPEND '(.)' ALL X
```

To prepend '---' to lines 10 through 20 of the file.

```
PREPEND '---' .10 .20
```

To prepend '[135]' to all non-excluded lines in the line range 3 through 200.

```
PREPEND '[135]' NX .3 .200
```

Comparison to CHANGE command

You can modify lines using **CHANGE** with pictures, in a manner similar to **PREPEND**.

If you wanted to rewrite the first example of **PREPEND '(.)' ALL X** as a **CHANGE**, it would look like this:

```
CHANGE P='=' F'(.=')(.)' ALL 1 1 X
```

The difference is that the **CHANGE** command will not work for zero-length lines; the search for **P='='** in column 1 fails on such lines. So, no zero-length lines would get modified. However, **PREPEND** inserts the string on every line, including zero-length lines. Depending on how you want to handle zero-length lines, both approaches can be useful.

Comparison to Power Typing

You can prepend text to a given line range by using the **PTYPE** command, setting Insert Mode on, moving the cursor to column 1, and typing the characters to be inserted at column 1. Power Typing has many powerful capabilities, but if you only wish to insert text at the beginning of lines, **PREPEND** may be simpler to use, and because it is a complete primary command, it can be brought back with the **RETRIEVE** commands and run again.

PRESERVE - Control Handling of Trailing Blanks

Syntax

PRESERVE	[<u>ON</u> OFF C]
-----------------	--------------------------------

Operands

ON | **OFF** | **C** The desired PRESERVE status

Description

The Profile option **PRESERVE** controls how SPFLite handles trailing spaces on text lines while writing the file to disk.

The value of **PRESERVE** is stored as part of the PROFILE options which are maintained individually by file type.

Preserve ON/OFF Handling

If **PRESERVE ON** is specified, trailing blanks are retained as-is and written to the file.

If **PRESERVE OFF** is specified, trailing blanks will be removed before writing the text to the file.

Preserve C Handling

If **PRESERVE C** is specified, trailing blanks are retained as-is and written to the file in the **same** way as **PRESERVE ON**, **except** when the last nonblank character of the line is a \ backslash. In this case **only**, such lines will have any trailing blanks removed before writing the text to the file. This option would be used in C and similar languages that use backslash as a continuation character but do not allow a backslash to be followed by whitespace.

(The language standards for C and C++, for example, do **not** allow the \ backslash continuation to be followed by whitespace, but require them to be the physically last character of the line.)

PRESERVE C ensures that the editor will not trim data lines (a file change detectable by programs like MAKE and DIFF) while guaranteeing that continuation lines do not contain spurious trailing blanks.

Timing of trailing-blank removal with PRESERVE OFF or PRESERVE C

The trimming action performed by PRESERVE OFF or PRESERVE C occurs when the file is saved or when END is processed, but during a SAVE operation when you continue editing the file, any existing trailing blanks are not removed from the then-current edit session. These trailing blanks will be removed when you close your file; this will be evident the next time you open your file again.

If you need these trailing blanks removed immediately, you should the **TR** line command. If you place a line command of **TR/** on line 1 of your file, the entire file will have trailing blanks removed from all lines.

PRINT - Send Selected Lines to the Printer

Syntax

```
PRINT      [ SETUP      |
            line-control-range
            [ X | NX ]
            [ U | NU ]
            [ NUM | NONUM ]
            ]
```

Operands

SETUP If **SETUP** is specified, it should be the only operand. It requests SPFLite to display a printer selection dialog to allow selection of your desired SPFLite default printer and printer font.

line-control-range The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

X | NX Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be examined, **NX** requests only non-excluded lines are to be examined. If neither **X** or **NX** are specified, all lines in the range will be examined.

U | NU Specifies a subset of the line range to be processed. **U** requests only User lines are to be processed, **NU** requests only non-User lines are to be processed. If neither **U** or **NU** are specified, all lines in the range will be processed.

NUM | NONUM Normally, the print output will **not** contain the line numbers on the left. Specifying **NUM** will cause these line numbers to be printed.

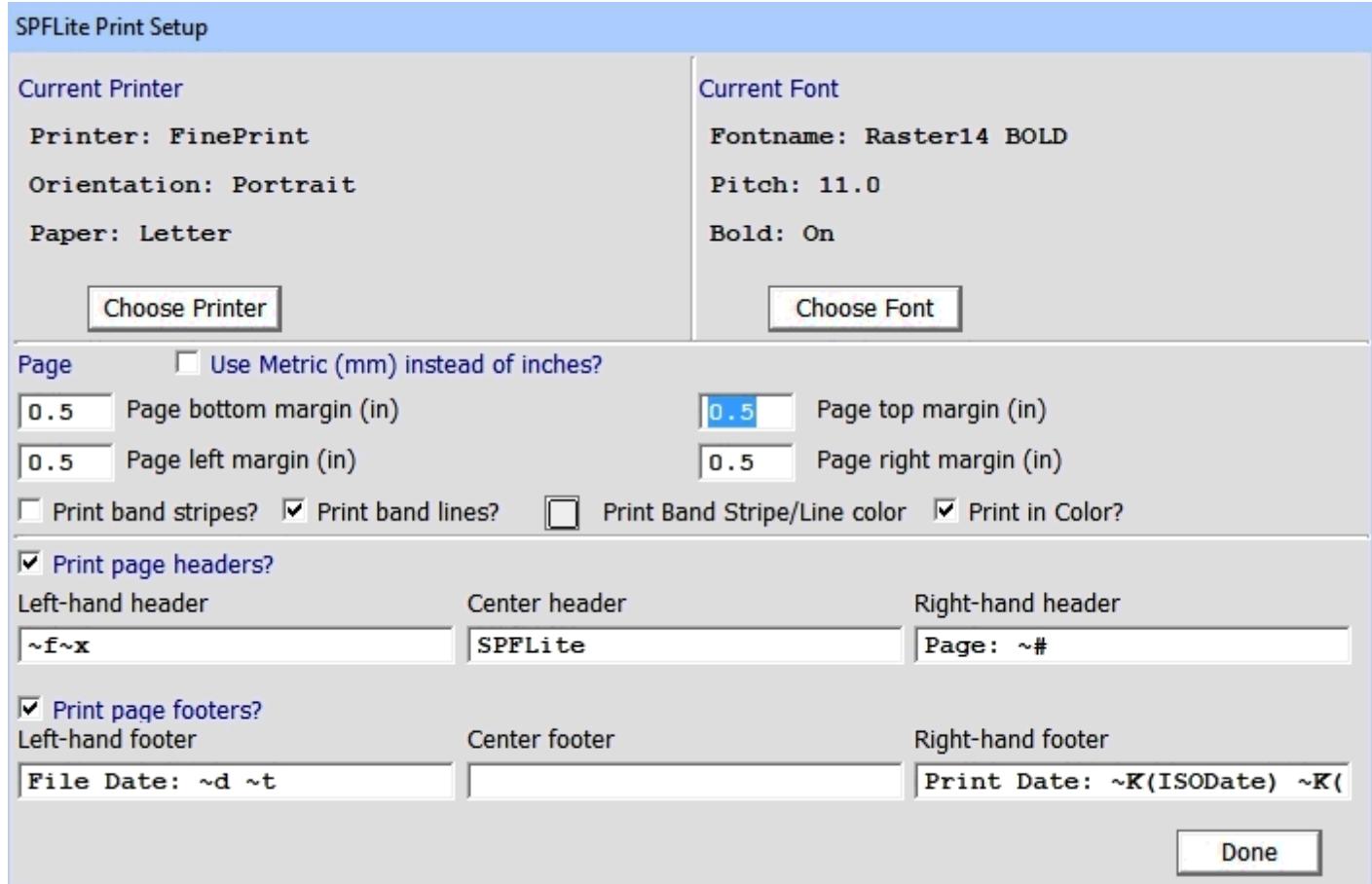
Description

PRINT directs a hard copy listing of the file being edited to your chosen default printer using the formatting parameters you specified during **PRINT SETUP** processing. You must specify a line range, either via command line operands (the line-control-range) or via line commands. See "[Line Control Range Specification](#)" for full details of the selection options available.

The listing is formatted without the line numbers of the file in the Left margin. If desired the keyword **NUM** may be entered to request the line numbers on the listing.

The PRINT SETUP dialog

When you issue the **PRINT SETUP** command, the following dialog will appear:



Printer Selection

The upper left portion shows the currently assigned printer and the chosen paper and orientation settings. If you wish to alter these settings, click the Choose Printer button and you will proceed to a standard Windows Printer selection dialog. Choose the printer and paper choices you desire and close the Printer selection dialog.

Font Selection

The upper right portion shows the currently assigned Font selection along with the Font pitch and Bold setting. If you wish to alter these settings, click the Choose Font button and you will proceed to a standard Windows Font selection dialog. Choose the font, pitch and bold choices you desire and close the Font selection dialog.

If you want to print in a font that matches the supplied **RASTER** screen font, you can use **RasterTTF**. **RasterTTF** is the Raster font converted to TrueType format. **RasterTTF** has the same character set as Raster, but is intended primarily for printing files. **RasterTTF** has the advantage of being scalable, while Raster and Raster14 are crisper fonts for the screen. To print a mainframe-style "listing" file as 66 lines x 132 columns, you can use RasterTTF 11 Pitch, 0.5 top margin, 0.4 bottom margin, in landscape mode; there will still be room for a header and footer on the page.

Page Settings

The middle section shows the current page settings. Your first choice here is to decide between measurements in Metric or English. Check the box labeled "Use Metric (mm) instead of inches?" if you wish to use Millimeters; otherwise leave unchecked to use inches.

Next, set the desired margin values into the 4 margin boxes.

Also located in this section are the options for Background banding. Two types of banding are available
Print Band Stripes This option, if activated, imitates the old striped continuous form paper commonly used

for line printers. The alternating horizontal stripes can assist greatly when reading source listings and reports.

Print Band Lines This option simply draws a horizontal line across the page every three lines.

Print Band Stripe/Line Color This box allows you to select the color to use for the Band Stripe/Line. Choose something which is not too bold and obtrusive, but which still benefits the output. You may have to experiment with your printer. When using Band Stripes, some printers (particularly laser printers) do not handle shaded areas well.

Print in color? If selected, and the file being edited has **HILITE AUTO ON** and has a valid **.AUTO** file, then the output sent to the printer will be done in the same colors as screen colorization. Otherwise, the output will be in normal uncolored text.

Page Headers and Footers

The bottom portion shows the page header and footer choices. Again, your first choice here is whether you want either of these to appear. There are two check boxes; one for Headers, one for footers. If you do not want either of these just clear their associated check box.

If you choose either or both of the header/footers to appear, then the left-hand header, center header and right-hand header fields come into play. (Same for the footer fields.)

Left-Hand Header/Footer

The data in this field will be left-justified in the Header/Footer line.

Center Header/Footer

The data in this field will be centered in the Header/Footer line.

Right-Hand Header/Footer

The data in this field will be right-justified in the Header/Footer line.

All of these fields are optional, and can be used in any combination. Note that if you enter very long strings and the page width is insufficient to 'fit' the three fields in the line, there will be some unpredictable overlaying of fields. Be aware of the sizes of these fields, particularly when using substitution variables for filenames, as these can be quite long.

Specifying the contents of headers and footers

Headers and footers can hold any desired text. Normal text placed here is left unmodified and is used as-is. However, the flexibility available comes into play when you use one or more of the many variables available which are dynamically substituted at print time to contain the current date, time, filename etc.

The following variables are available for use. Note that variables indicated by a leading ~ tilde are substituted as-is with alphabetic characters unmodified, while those indicated by a leading ^ caret are substituted with alphabetic characters set to upper case.

~# or ^#	The current printed page number.
~d or ^d	The current file date as yyyy-mm-dd.
~f or ^f	The current base filename. For C:\MYDATA.TXT it would be MYDATA
~n or ^n	The current full filename including path.
~p or ^p	The current file path.

~t or ^t	The current file time as hh:mm:ss.
~x or ^x	The current file extension. For MYDATA.TXT it would be TXT
~k (keyname) or ^k (keyname)	The text returned from a keyboard definition where 'keyname' is the key name or keyboard primitive. This uses the new ~K() macro ability see the full description of these at "Working with ---"

When you have completed your setup, click on the Done button. Your print settings will be saved and used for all subsequent PRINT functions.

Example

If you look back at the sample image above showing the **PRINT SETUP** dialog, the page headers and footers created when printing the file **C:\Data Dir\Source.ASM** using those parameters would appear as:

```
-----10-----20-----30-----40-----50-----60-----70
Header> Source.ASM           SPFLite           Page: 1
000001 Source lines
000002 Source lines
000003 Source lines
.
.
.
Footer> File Date: 2012-01-13 09:10:34      Print Date: 2012-02-01 13:12:21
```

Note: The full text for the Right-Hand Footer is not visible in the image but contained:

Print Date: ~K(ISODATE) ~K(ISOTIME)

PROFILE - Display Current File Profile Variables

Syntax

```
PROFILE      [ { LOCK | UNLOCK }           |
                { RESET }                  |
                { COPY profile-name }      |
                { USE  profile-name }      |
                { NEW  profile-name }      |
                { EDIT [ profile-name ] } ]
```

Operands

LOCK	Requests the profile be LOCKED. When locked, all changes to the profile will be in effect only for the duration of the current edit session; and are not written to permanent profile storage.
UNLOCK	Requests the profile be UNLOCKED. When unlocked, changes to profile variables are saved to permanent storage and will be used in all future edits of this file type.
	The permanent storage for a profile is a text file in the SPFLite\Profiles directory, with a name of <i>profile-name.INI</i>
RESET	Requests all profile variables be reset to the SPFLite standard values (described below)
COPY profile-name	Requests all profile variables be copied from another existing file profile.
USE profile-name	Requests this profile to use another specified profile's values as its own. This is particularly useful in some programming languages which use a variety of file types to store data, but whose editing characteristics are identical. (An example is the C language that uses .C for programs and .H for header files, both of which are formatted and handled the same way.) It allows multiple file types to 'share' a profile so that a change to a variable need only be done once to have it take effect over all the shared file types. The keyword USE may also be specified as USING .
NEW profile-name	Some of the profile variables (such as LRECL , SOURCE and EOL) affect how SPFLite loads a file into the edit work space. If the file type profile has the incorrect values, the data will not load correctly. Thus, for the 1st time edit of the file, you need the profile to have already been created. NEW provides this ability; see below.
EDIT profile-name	Will bring up the Profile edit dialog for an existing Profile. If several changes to profile settings are going to be made at the same time, it may be easier to make these changes using the dialog rather than doing it with a series of primary commands.
	Note: If no profile-name is entered, EDIT will open the dialog for the currently active profile.

Abbreviations and Aliases

PROFILE can also be spelled as **PRO** or **PROF**

LOCK can also be spelled as **LOCKED**

UNLOCK can also be spelled as **UNLOCKED**

USE can also be spelled as **USING**

Description

If the **PROFILE** command is entered without any operands, it will cause SPFLite to insert a series of **=PROF>** lines into the text display showing the current status of all Profile settings for the current file type. Special lines for **=COLS>**, **=WORD>**, **=TABS>**, **=BNDS>** and **=MARK>** will also be displayed, since these settings are part of the Profile.

Here is a sample display that will be produced by using **PROFILE** without operands:

The **PROFILE LOCK** and **PROFILE UNLOCK** commands alter the LOCK status of a profile. When a profile is locked, any changes made are in effect for that edit session only and are not saved.

PROFILE RESET can be used if you want to discard all modifications you have made and start over with a standard default set of Profile options.

Note that **PROFILE** commands that contain a *profile-name* operand are order dependent. That is,

- the command **PROFILE USE TXT** is legal, but
 - the command **PROFILE TXT USE** is illegal.

This is done so that any name can be used for a profile name, including keywords from the **PROFILE** command.

For example, the command **PROFILE USE USE** is **legal**. The first **USE** is a keyword designating a **PROFILE USE** command, and the second **USE** is the profile's **name** which corresponds to the file's type, and just so happens to be spelled the same as a keyword.

Creating a Profile before needed

The **PROFILE NEW** profile-name option allows you to set up a profile before its first use. This is mandatory when creating profiles for files such as EBCDIC mainframe files, or other non-standard formats. To create such a profile ahead of time, do the following:

- On the command line enter **EDTT NEW** to open a new tab on an (Empty) file.

- Enter **PROFILE NEW profile-name**. Here, **profile-name** is the name of the file extension for which you want to create as a profile.
- A separate pop-up dialog will open where you may select the particular options you require. Click on Done when you are finished.
- At this point, you might also want to customize other settings like WORD and TABS.
- **CANCEL** the dummy edit tab you are using, because the Edit session was opened only for the side-effect of creating a Profile, not to create the data file itself.

Your new profile is ready to use.

PROFILE EDIT profile-name

Brings up the SPFLite Profile Editor window on the named profile. Only existing profiles can be edited. Attempting to edit an undefined profile causes the message “Specified Profile name does not exist” to be issued. Either use the correct name, or perform a **PROFILE NEW** command instead to create a new Profile. **PROFILE EDIT** may be issued from an edit window or from the File Manager, thus a file of the given file type (the same as the profile name) need not be open to do a **PROFILE EDIT** command.

Default Profile values set by a PROFILE RESET command

Profile	UNLOCK
AUTOBKUP	OFF
AUTOSAVE	OFF NOPROMPT
CAPS	OFF
CASE	T
COLS	OFF
EOL	CRLF
HEX	OFF
HILITE	ON FIND AUTO
LRECL	0
MARK	ON
MINLEN	0
PRESERVE	ON
RECFM	U
SCROLL	CSR
SETUNDO	10
SOURCE	ANSI
START	OFF
STATE	OFF
TABS	OFF
XTABS	8
USING	(null)
BNDS line	1 MAX
WORD line	A-Z a-z 0-9
MARK line	(none)
TABS line	(none)

PTYPE - Enter PowerType Mode

Syntax

PTYPE	[line-control-range]
	[X NX]
	[U NU]

Operands

line-control-range	The range of lines which are to be processed by the PTYPE command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
	A line range can also be specified by a C/CC block. Additionally, for the PTYPE command, a WMM block will be treated as equivalent to a C/CC block; that is, if you begin power-type using lines marked by a M/MM block, the data will be modified as usual, rather than being deleted afterwards, which is usually done in a M/MM block.
	If a line range is omitted from PTYPE , you must use an X NX option and/or define a line range with a C/CC block. If you issue a PTYPE command with neither the line range or an X NX option, it will not work, but you will get a Pending line range message instead.
	To Power Type over the entire edit file, you can issue a RESET to unexclude all the lines and then say PTYPE NX , or you can put explicit line labels such as PTYPE .ZFIRST .ZLAST , which can be abbreviated to PT .ZF .ZL , or you could place the line command C/ on line 1 of the file.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed by Power Type, and NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines specified in the range are to be processed.
	If the first line being modified by PTYPE is excluded, the first thing PTYPE will do is unexclude that line. (If that were not done, you would not be able to see the line you were modifying, which would be confusing and hard to work with.)
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.

Abbreviations and Aliases

PTYPE can also be spelled as **PT**

Description

The **PTYPE** command begins a Power Typing session using the lines you specify in the line-range operand or **C/CC** block, which can be limited to just excluded (**X**) or just non-excluded (**NX**) lines if you wish. You can also

limit PTYPE to User lines (**U**) or non-User lines (**NU**).

Power Typing means that for each character you type, or each keyboard primitive function you use, the action is applied to every line in the Power Typing line range, in parallel, at the same time.

Some editors refer to this capability as “column-mode editing” because the same action is applied at the same column position of every line at the same time, in parallel. You will generally find the editing capabilities of SPFLite's Power Typing to be more powerful than the “column mode” features of other editors, because the line range, **X|NX** and **U|NU** options allow for the selection of non-contiguous lines, and because of the wide range of keyboard functions you can use while within Power Typing mode.

Once Power Typing mode begins, the edit display moves to the first line of the selected line range, in the same way a **LOCATE** command would do. You will see the message, **Entering Power Type mode, Press Enter to exit**, and the status line will show **PowerType**.

The cursor is then moved to column 1 of that first line, which then acts as a ‘prototype’ or ‘model line’. As you enter characters keys or invoke keyboard functions, you will see the effects reflected on the model line, and on every other line that is included in the Power Typing line range.

Power Typing mode remains in effect until you press Enter. Once that is done, the message **Entering Power Type mode, Press Enter to exit** will disappear, and the status line indicator showing **PowerType** will be removed as well.

Note: Power Typing mode is allowed while within a multi-edit session.

See the article [Working with Power Typing Mode](#) for more information.

QUERY - Display a Profile Setting

Syntax

QUERY Q	Profile-setting
-------------------------	------------------------

Operands

Profile-setting One of the following keywords may be entered. QUERY will respond with a simple message providing the current setting for that Profile value.

ACTION	AUTOBKUP	AUTOCAPS	AUTOSAVE
BOUNDS	CAPS	CASE	CHANGE
COLLATE	COLS	ENUMWITH	EOL
FOLD	GLUEWITH	HEX	HIDE
HILITE	LRECL	MARK	MINLEN
NOTIFY	PAGE	PRESERVE	RECFM
SETUNDO	SOURCE	START	STATE
SUBARG	SUBCMD	TABS	XTABS

Description

The **QUERY** command simply generates a small message containing the current setting for the request Profile variable. It is displayed in the standard message area at the upper right of the screen.

For Example, **Q AUTOSAVE** might display the message **AUTOSAVE set to OFF PROMPT**

RCHANGE - Repeat Change

Syntax

```
RCHANGE
```

Operands

None

Description

RCHANGE repeats the change requested by the most recent [CHANGE](#) command. You can use this command to repeatedly change other occurrences of the search string.

Note: **RCHANGE** is normally directly assigned to a defined keyboard key, although you can issue it directly from the Command line. Traditionally, the **RCHANGE** command is mapped to F6.

RCHANGE will repeat the "changing action" requested by the most recent **DELETE SPLIT** or **JOIN** primary command.

See [Working with SPLIT and JOIN Commands](#) for more information.

RECALL - Recall (Open) a Favorite File List

Syntax

RECALL | **RC**

[**RECENT** | **list-name**]

Operands

list-name

Specifies the name of a File List to be displayed. The *list-name* operand is treated as case-insensitive.

- When *list-name* is omitted, a default of **RECENT** is assumed. **RECALL RECENT** causes the **Recent Files** File List to be displayed.
- If *list-name* is **FAV**, **FAVORITE** or **FAVOURITE**, it cause the **Favorite Files** File List to be displayed.
- If *list-name* is **FOUND**, it cause the **Found Files** File List to be displayed.
- If *list-name* is **OPEN**, it cause the **Open Files** File List to be displayed. **RECALL OPEN** and [**SWAP LIST**](#) perform the same function.
- If *list-name* is **PATHS**, it cause the **Recent Paths** File List to be displayed.
- If *list-name* is anything else, like **MyList**, it causes the **Named Favorites** File List **MyList.FLIST** to be displayed.

Description

RECALL is a primary command that may be issued from any Edit or Browse tab, from a Clipboard edit tab, from a SET-variable edit tab, or from the File Manager command line.

RECALL will switch the display to the File Manager, if not already there, and then displays the list of files in the **Recent Files** File List, or another named File List if *name* is specified. This action is equivalent to clicking on the File Manager tab and then clicking on the line that says **Recent Files** File List, or clicking on **Named Favorites** and then clicking on *name.FILELIST*.

Note: Former Tritus SPF users will recognize the **RECALL** | **RC** command as performing a similar function

Reserved File List names

The File List names **FOUND**, **OPEN** and **PATHS** are reserved for displaying the special lists described above. This is in addition to **RECENT**, **FAV**, **FAVORITE** and **FAVOURITE** which were already reserved.

Reserved File List names cannot be used for user-defined named favorites for the [**FAVORITE**](#) and [**MAKELIST**](#) commands. Even though you can issue a command like **RECALL OPEN**, you cannot say **FAVORITE OPEN**, because that would imply you were creating (or adding a file name to) a named favorite File List called **OPEN**, which SPFLite has reserved for its internal list of currently-open files.

RECFM - Set Record Format

Syntax

RECFM	[{ <u>U</u> F V VBI VLI }]
-------	--

Operands

U	The file uses Undefined type records. i.e. The records are separated by unique delimiters which should be specified using the EOL command.
F	The file uses Fixed record length records. Fixed records may or may not also use unique delimiters. If delimiters are also used, the specific delimiters should be set via the EOL command. As well, the length of the fixed records should be set via the LRECL command.
V	The file uses Variable record format. Typically this means is originated at an IBM mainframe site or from an emulator (such as Hercules). If so, then EBCDIC would probably also need setting via the SOURCE command. Note this support is only for unblocked variable, SPFLite does not support VB or VBS record formats.
VBI	The file uses variable format where a 4 byte RDW (Record Descriptor Word) precedes the record data. The RDW contains the length of the data, not including the RDW itself. VBI indicates the word is in ' Big -endian' format.
VLI	The file uses variable format where a 4 byte RDW (Record Descriptor Word) precedes the record data. The RDW contains the length of the data, not including the RDW itself. VLI indicates the word is in ' Little -endian' format.

Description

If entered without any operand, RECFM will display the current setting.

In order to properly read an input file, SPFLite must be provided with some information regarding what the file is expected. Basic to this is the Record Format.

More details on this area can be found in [Handling Non-Windows Text Files](#)

The value is stored as part of the PROFILE options which are maintained individually by file type.

Note that **RECFM** is an IBM mainframe acronym, meaning **Record Format**.

REDO - Redo an UNDO Action

Syntax

```
REDO
```

Operands

None

Description

Sometimes when you repeat an [UNDO](#) to reach a previous edit status, you might **UNDO** too much, especially if **UNDO** is mapped to a key that auto-repeats. **REDO** allows you to reapply changes, effectively undoing the **UNDO**.

The **REDO** command is a complement to **UNDO** and will successively reapply the changes undone by the **UNDO** command.

You can only REDO as many times as successive **UNDO** commands have been done. The maximum number of **UNDO** operations is set by the [SETUNDO](#) command, which has an upper limit of 25, and is set on a PROFILE basis.

RELOAD - Reload Current Edit File

Syntax

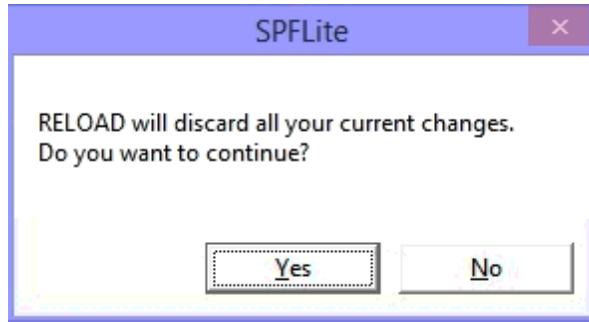
```
RELOAD
```

Operands

None

Description

The **RELOAD** command will discard any unsaved changes in the current edit file and then reload the file from its current contents on disk. If any unsaved changes exist, a popup will ask for confirmation of the Reload:



RELOAD may be issued from an Edit or Browse session.

If you are editing the contents of the Windows clipboard in a Clipboard edit session, you will not be notified if another application alters the contents of the clipboard. You cannot issue the **RELOAD** command from a Clipboard edit session.

RENAME - Rename the Current Edit File

Syntax

RENAME	[file-name]
---------------	----------------------

Operands

file-name

If you specify a simple (unqualified) file-name, then the file will be in the same directory as the file being edited after being renamed. If you used a relative file-path (such as MYPATH\myfile.TXT) the file will be located relative to the directory where it is currently being edited. If the file-name is a fully-qualified name, it will be located directly where you specify. Note that if you rename a file to anything other than a simple name, the file will be removed from its current directory location and relocated to the new directory you specify.

If you do not specify a file name, a conventional Windows save dialog will be presented to allow you to select the new name the file is to be known as, and optionally the new directory where it is to be located.

Description

The **RENAME** primary command allows the currently-edited file to be renamed while being edited. If the new name also introduces a previously unreferenced file extension, the Profile for the new extension will be created based on the old file's Profile.

When the command has no argument, **RENAME** will cause a Rename popup to be displayed, which asks for a new file name. Once the file has been renamed, you may resume editing the file as usual. You can **Cancel** the Rename popup if you change your mind and don't want to rename it.

RENAME may be issued from either an Edit session or a Browse session.

If you **RENAME** a browsed file, it does not change its content, but does change its name. If you opened the file in Browse mode because another application was accessing (and possibly updating) this file, renaming it could interfere with the other application, and so this should be done with care.

RENUM - Evenly renumber lines in file

Syntax

```
RENUM
```

Operands

RENUM has no operands

Abbreviations and Aliases

RENUM can also be spelled as **REN** and **RENUMBER**

Description

The **RENUM** command will completely assign new sequence numbers. The starting number and increment are based on the number of lines in the file, and the specified size of the sequence number field. The following table summarizes this.

File Size	Sequence # Size	Starting Number	Increment
Under 1000	4	0010	10
	5	00100	100
	6	000100	100
	7	0000100	100
	8	00000100	100
Under 10000	4	0001	1
	5	00010	10
	6	000100	100
	7	0000100	100
	8	00000100	100
Under 100000	4	0001	1
	5	00001	1
	6	000010	10
	7	0000100	100
	8	00000100	100
Under 1000000	4	0001	1
	5	00001	1
	6	000001	1
	7	0000010	10
	8	000000100	100

All numbering commands replace data in existing column positions, with no attempt to save or shift-over your

data to make room for these numbers. It is an "overlay operation", not an "insert operation". So, for instance, if you use COBOL numbering, columns 1-6 of every line of your file will be replaced by sequence numbers, **regardless of any data that may already be present there**. It is your responsibility to alter the contents of your file, if necessary, if that is not the outcome you wanted.

See [Working with Sequence Numbering](#) for more information.

Release note for SPFLite version 8.5

In SPFLite prior to release 8.5, the editor primary command **REN** was an abbreviation for the command **RENAME**. As of SPFLite version 8.5, the editor primary command **REN** is now an abbreviation for the sequence numbering command **RENUM**. This change was made for ISPF compatibility.

Because of this change, the **RENAME** command no longer has any abbreviations. You must fully spell out the editor primary command name **RENAME**.

REPLACE - Replace a File

Syntax

```
REPLACE      { line-control-range }
                [ filename ]
                [ X | NX ]
                [ U | NU ]
```

Operands

line-control-range The range of lines which are to be included by the command. Line control ranges provide a powerful tool to customize the range of lines to be included. The full syntax and allowable operands which make up a line control range is discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

Note: Prior editions of this document showed syntax that implied that the line-control range on **REPLACE** was optional. That is not correct. You must specify a line-control-range, **unless** the line-control-range is implied via a **C/CC** or **M/MM** block. We are trying to emphasize that fact by showing the syntax in **red** above, and in braces rather than brackets.

file-name The name of the file you wish to replace.

X | **NX** Specifies a subset of the line range to be included. **X** requests only excluded lines are to be included, **NX** requests only non-excluded lines are to be included. If neither **X** or **NX** are specified, all lines in the range will be included.

U | **NU** Specifies a subset of the line range to be included. **U** requests only User lines are to be included, **NU** requests only non-User lines are to be included. If neither **U** or **NU** are specified, all lines in the range will be included.

Description

REPLACE will use all or a specified portion of the Edit data to replace a new external file. If you do not specify a full pathname as part of file-name, then the file will be created in the same directory as the file being edited. Note: If the filename does not already exists, then command will create it as a new file. The [CREATE](#) and **REPLACE** commands are almost identical, except [CREATE](#) prevents overwriting of an existing file, whereas **REPLACE** does not.

To copy the entire Edit data into an existing file:

On the Command line, type:

```
REPLACE file-name
```

The filename operand is optional. If you do not specify it, the standard Windows File Open dialog will be displayed. Enter the filename in this dialog.

Press Enter. The entire Edit data will replace the existing contents of the filename.

To copy only a portion of the Edit data into an existing file:

On the Command line, type:

REPLACE filename line-control-range

The filename operand is optional. If you do not specify it the standard Windows File Open dialog will be displayed. Enter the filename in this dialog.

The line-control-range would typically specify the starting and ending line numbers to be included in the replaced file (or some other line-control-range variation). As described under "[Line Control Range Specification](#)" a variety of other possibilities for specifying the line range exist including marking the lines with the [CC/CC](#) or [MM/MM](#) line commands.

Press Enter. The specified Edit data is copied/moved to replace the specified filename.

File Format

If you wish to write a file in a format other than the normal default specified by your **PROFILE EOL** and **SOURCE** settings see ["Handling Non-Windows Text Files"](#) for additional info.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the **REPLACE** command, the default directory used for writing the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the path for that active file is used as the default for the command.
- If there is **no** active file [when the tab header displays (Empty)], the currently displayed directory of File Manager will be used.

RESET - Reset (Edit Mode)

Syntax

```
RESET [ line-control-range ]
      [ ALL ]
      [ CHANGE ]
      [ COMMAND ]
      [ WORD ]
      [ EXCLUDED ]
      [ FIND ]
      [ USER ]
      [ HIDE ]
      [ LABEL ]
      [ RETRIEVE ]
      [ SPECIAL ]
      [ STATE ]
      [ TAGS ]
      [ colorname ]
```

Operands

line-control-range The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#). Refer to that section of the documentation for details.

Note: Normally the line-control-range can be specified via the command line **or** via **C/CC** line commands. For **RESET**, the use of the **C/CC** option is not allowed; you must use a line-control-range operand.

ALL If entered with **none** of the following operands, **ALL** requests a reset of **all** the options below except **RETRIEVE**, **LABELS** and **TAGS** and the names of the colors. (See note below.) . If entered along with any of those operands, it has no special effect

COMMAND Clear all pending line commands from the specified range.

WORD Reset the **WORD** values back to the SPFLite default.

EXCLUDED Clear all excluded lines back to non-excluded status.

FIND Clear all Hi-lighting created by a prior FIND/CHANGE type command

USER Reverts all User Lines ("U lines") back to ordinary lines ("V lines").

Note that U lines have a | vertical in the "gap column", the space between the line number and the actual text, to mark them as User Lines. Resetting U lines back to ordinary V lines will cause this gap column to be displayed as blank again.

HIDE Turn HIDE mode off. **RESET HIDE** is equivalent to a **HIDE OFF** command.

LABEL	Clear all current labels markers from the specified range.
RETRIEVE	Reset the primary command Retrieve stack.
SPECIAL	Delete all special lines (COLS , WORD , TABS etc.) from the specified range.
STATE	Clear all STATE related values from the specified range. This is equivalent to issuing the three operands EXCLUDED , LABELS and TAGS .
TAGS	Clear all tags from the specified range.
colorname	Clears the specified color from all lines of text within the line range, returning text that was in the specified color back to the "standard" text color. Valid color names are (BLUE , GREEN , YELLOW , RED , BLACK , NAVY , TEAL , VIOLET , ORANGE , GRAY , LIME , CYAN , PINK , MAGENTA and WHITE).
COLOR	Removes all of the colors from all lines of text within the line range, returning all text back to the "standard" text color.

Abbreviations and Aliases

RESET can also be spelled as **RES**

CHANGE can also be spelled as **C** or **CHG**

COMMAND can also be spelled as **CMD** or **COM**

EXCLUDED can also be spelled as **X**, **EX**, **EXC**, or **EXCLUDE**

LABEL can also be spelled as **LAB** or **LABELS**

SPECIAL can also be spelled as **SPE**

TAGS can also be spelled as **TAG**

USER can also be spelled as **U**

WORDS can also be spelled as **WORD**

Description

The **RESET** command allows you to clear or reset one or more types of line status.

If **RESET** is entered with no operands, it acts as if you had entered **all** operands except **RETRIEVE**, **LABELS** and **TAGS** and the names of the colors. (See note below.)

That is, a **RESET** by itself will not clear labels or tags, nor will it remove the colors set by the virtual highlighting-pen keyboard functions.

The ISPF command **RESET ERROR** is not supported in SPFLite; use of the keyword **ERROR** is invalid.

Note:

- the effect of **RESET X** can also be achieved with **SHOW ALL**
- the effect of **RESET U** can also be achieved with **REVERT ALL (VV ALL)**

Special handling of **RESET** when resetting User Line status

Long time ISPF users are accustomed to **RESET** performing a number of default resetting actions, when no options are specified. A plain **RESET** is commonly used to reset excluded lines, as if **RESET X** were specified. These default actions can be extended to the new User Line feature. That is, a plain **RESET** can revert any existing User Lines back to ordinary non-User Lines, as if **RESET U** were specified.

However, since User Lines are a new facility, there is no prior experience to form a consensus about how this should best be handled. Some users may want User Lines implicitly reverted, and others might not.

To address this, a plain **RESET** will revert any existing User Lines back to ordinary non-User Lines, but **only** if you enable the checkbox on the **General Options** dialog that says, "**Default RESET will revert user line status**". By default, this checkbox is disabled.

See [Options - General](#) for more information.

RESET - Reset (File Manager Mode)

Syntax

RESET [ALL]
[EXCLUDE] or X, EX, EXCL
[FORGET] or F, FOR, FORGOT

Operands

ALL Will reset all Excluded and Forgotten file names.

EXCLUDE Will reset all Excluded file names

FORGET Will reset all Forgotten file names.

Description

The **RESET** command allows you to reverse the effect of EXCLUDE and FORGET commands used to temporarily hide files from the File Manager display.

RETF - Retrieve Forward Direction

Syntax

```
RETF
```

Operands

None

Description

RETF retrieves commands from the command stack moving in the direction from the oldest command in the command stack toward the most recent commands in the command stack, in the opposite direction done by **RETRIEVE**.

Forward retrieve (**RETF**) retrieves the oldest command on the command stack, if **RETF** is entered immediately after a command is executed, before performing a **RETRIEVE**. That is, the list of saved commands that can be retrieved is a circular list.

So, if **RETF** is a retrieve forwards, then a regular **RETRIEVE** is a retrieve backwards - that is, a command retrieval going backwards in time starting at the most recently issued command.

The Editor keeps the last 50 unique command lines available for retrieval. The list of previous commands is persistent, and will be saved and restored from one SPFLite session to the next.

There is only one list of saved commands, regardless of the number of file tabs currently open. So, when a command is retrieved, there is no connection between a command, the particular file tab where the command was originally issued, or the tab where the command is retrieved.

See also the [RETRIEVE - Retrieve Previous Commands](#) command.

See also [Options - General](#) for setting the minimum Retrieve length.

RETRIEVE - Retrieve Previous Commands

Syntax

```
RETRIEVE
```

Operands

None

Description

The **RETRIEVE** command will bring back to the Command line the previously issued command for use to modify and re-enter. Successive **RETRIEVE** commands will retrieve the previous, next previous, etc. commands. The Editor keeps the last 50 unique command lines available for retrieval. Note that the list of previous commands will be saved and recalled between SPFLite sessions.

There is only one list of saved commands, regardless of the number of file tabs currently open. So, when a command is retrieved, there is no connection between a command, the particular file tab where the command was originally issued, or the tab where the command is retrieved.

See also the [RETF - Retrieve Forward Direction](#) command.

See also [Options - General](#) for setting the minimum Retrieve length.

REVERT - Revert User Line to Ordinary Line

Syntax

```
REVERT      [string]
            [ CHAR | WORD | PREFIX | SUFFIX ]
            [ start-column [ end-column ] ]
            [ line-control-range ]
            [ color-selection-criteria ]
            [ MX | DX ]
            [ X | NX ]
            [ ALL ]
            [ TOP ]
```

Operands

string	The string operand is optional. If provided, it requests a search, like FIND, to locate the lines to be un-marked as User lines. If not provided, then the line-control-range specification will be used to select lines.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-Word characters.
PREFIX	Locates search-string at the beginning of a word.
SUFFIX	Locates search-string at the end of a word
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or

NX are specified, all lines in the range will be eligible to be processed.

MX	MX requests that all lines which DO contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

REVERT can also be spelled as **VV**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

REVERT will remove ("revert") the User Line status from all lines which meet the specified criteria. This is an alternative method to using the **V** / **VV** line commands to remove the User Line status of selected lines.

When a U line reverts to an ordinary V line, the **|** vertical bar that marks the "gap column" will disappear.

Example uses of the REVERT command

To revert all User Lines from label **.FROM** to label **.TO** back to "ordinary" V lines:

```
REVERT ALL .FROM .TO
```

To revert all User Lines containing the string **TWO** back to "ordinary" V lines:

```
REVERT ALL "TWO"
```

To revert all User Lines containing the word **TEST** between columns 5 and 10, and which are not highlighted in **BLUE**, back to "ordinary" V lines:

```
REVERT "TEST" WORD 5 10 -BLUE ALL
```

For more information on User lines see "[Working with User lines](#)"

RFIND - Repeat Previous Find Command

Syntax

RFIND	[REVERSE]
-------	-------------

Operands

REVERSE	Causes the search direction to proceed opposite of the original FIND or NFIND command. If the original direction was forwards (due to a FIRST or NEXT operand, or by an implied NEXT operand) RFIND REVERSE will search backwards. If the original direction was backwards (due to a LAST or PREV operand) RFIND REVERSE will search forwards.
----------------	---

Description

RFIND repeats the find requested by the most recent [FIND](#) or [NFIND](#) command.

You can use this command to repeatedly find other occurrences of the search string. Note: **RFIND** is normally directly assigned to a defined keyboard key, although you can issue it directly from the command line.

RFIND is traditionally mapped to F5. You may find it convenient to map **RFIND REVERSE** to F5 with a modifier key, such as Shift-F5, though any desired key may be chosen.

The intent of the **REVERSE** option is to allow you to start a **FIND** operation in one direction, and then "change your mind" and search in the opposite direction, say, in case you passed up a line of interest and now you wish to 'go back' and find it again.

The **RFIND** command works the same way as it does in IBM ISPF. Many users will find the new SPFLite command **RLOCFIND** to be more useful than **RFIND**. See [RLOCFIND - Repeat Previous Find or Locate Command](#) for more information.

RFIND will repeat the "finding action" requested by the most recent **DELETE**, **SPLIT** or **JOIN** primary command. See [Working with SPLIT and JOIN Commands](#) for more information.

RLOC - Repeat Previous Locate Command

Syntax

RLOC	[REVERSE]
------	-------------

Operands

REVERSE	Causes the search direction to proceed opposite of the original LOCATE command. If the original direction was forwards (due to a FIRST or NEXT operand, or by an implied NEXT operand) RLOC REVERSE will search backwards. If the original direction was backwards (due to a LAST or PREV operand) RLOC REVERSE will search forwards.
----------------	---

Description

RLOC repeats the locate requested by the most recent [LOCATE](#) command.

You can use this command to repeatedly locate other occurrences of the locate request. Note: **RLOC** is normally directly assigned to a defined keyboard key, although you can issue it directly from the command line.

You may find it convenient to map **RLOC** to a function key, and **RLOC REVERSE** to the same function key with a modifier key, such as Shift, though any desired keys may be chosen.

The intent of the **REVERSE** option is to allow you to start a **LOCATE** operation in one direction, and then "change your mind" and search in the opposite direction, say, in case you passed up a line of interest and now you wish to 'go back' and locate it again.

The **RLOC** command works the same way as it does in IBM ISPF. Many users will find the new SPFLite command **RLOCFIND** to be more useful than **RLOC**. See [RLOCFIND - Repeat Previous Find or Locate Command](#) for more information.

RLOCFIND - Repeat Previous Find or Locate Command

Syntax

RLOCFIND	[REVERSE]
-----------------	--------------------

Operands

REVERSE	Causes the search direction to proceed opposite of the original LOCATE command. If the original direction was forwards (due to a FIRST or NEXT operand, or by an implied NEXT operand) RLOCFIND REVERSE will search backwards. If the original direction was backwards (due to a LAST or PREV operand) RLOCFIND REVERSE will search forwards .
----------------	--

Description

RLOCFIND is a hybrid combination of the [RLOC](#) and [RFIND](#) commands. Since the **RLOC** and **RFIND** commands are typically assigned to a defined keyboard key, the **RLOCFIND** allows a single key to serve the function of separate **RLOC** and **RFIND** keys. **RLOCFIND** is an extension to ISPF, which does not have this command.

It performs this by repeating the most previous **FIND** or **LOCATE** command issued. If the most recent command issued was a **FIND/RFIND**, then **RLOCFIND** acts like an **RFIND** command. If the most recent command issued was a **LOCATE/RLOC**, then **RLOCFIND** acts like an **RLOC** command.

As with **RFIND** and **RLOC** commands, the **RLOCFIND** command will normally be directly assigned to a defined keyboard key, although you can issue it directly from the command line.

You may find it convenient to map **RLOCFIND** to F5 and **RLOCFIND REVERSE** to F5 with a modifier key, such as Shift-F5, though any desired keys may be chosen.

The intent of the **REVERSE** option is to allow you to start a **LOCATE** or **FIND** operation in one direction, and then "change your mind" and search in the opposite direction, say, in case you passed up a line of interest and now you wish to 'go back' and find it again.

RLOCFIND will repeat the "finding action" requested by the most recent **DELETE**, **SPLIT** or **JOIN** primary command.

See [Working with SPLIT and JOIN Commands](#) for more information.

RUN - Execute the current Script

Syntax

RUN	[script-operands]
------------	----------------------------

Operands

script- operands	Optional operands to be passed to the script
-----------------------------	--

Description

The **RUN** command will save the current edit session to the \RUN sub-folder in the SPFLite data directory. It will then issue the command to start the named script, including any optional operands specified on the **RUN** command itself. To properly operate, the file type of the current edit file, must be registered in the system and associated with its appropriate script processor.

For this reason, **RUN** will only operate on edit sessions where the file is has a single "known name". It will not function in Clip, Set-Edit, MEdit or Cloned sessions which have no single associated file name.

A given script is executed as usual, being handled by the command handler associated with files of the given type: BAT files are processed directly by CMD.EXE, REXX scripts are executed by REXX.EXE (for instance) etc. just as they would if launched from a Windows command prompt.

Note: For specialized scripts, such as REXX or Perl, you may have to configure Windows so it knows what command processor or script interpreter to use when running your particular script. At a Windows command prompt, issue the commands **HELP FTYPE** and **HELP ASSOC** for more information. You may also need or wish to modify the contents of the **PATHEXT** system environment variable.

The RUN command saves a temporary copy of the script for execution purposes, but does **not** save the file in the edit session. So, you can RUN scripts containing unsaved changes without having to save them first. The concept is the same as how SUBMIT can be used to submit jobs that may contain unsaved data changes.

File Cleanup

Because the files are temporarily saved in the SPFLite\RUN folder, SPFLite will automatically purge this folder of files greater than one day old, to prevent a buildup of these temporary files. This way, you are not required to maintain these folders yourself.

Command Window Closing

You may control whether the command window opened by the **RUN** command remains open at the completion of the command. There is an option under [Options => Submit](#) which allows you to specify your choice. In the RUN parameters box on the Submit tab enter **/C** to close the window on completion, or **/K** to keep the window open.

SAVE - Save Data and Continue Edit

Syntax

```
SAVE
```

Operands

None

Description

SAVE writes the data to the same file from which it was retrieved.

If you wish to replace the data in another file, or create an entirely new file, you may use the [REPLACE](#), [SAVEAS](#) or [CREATE](#) commands respectively.

When a **SAVE** command is issued for a newly created file that does not yet have a name (you will know this by the file-tab label of **(Empty)** for this file), you will see the same file-saving dialog that you will see when **SAVEAS** is used. A window will appear with a title of **Specify file to SaveAs**, where you can enter the new file's name. The **SAVEAS** command can still be used as previously to achieve the same effect.

The advantage of this change is that if you had a key mapped to the **SAVE** command (such as Ctrl-S, for instance) that same key could be used to save both new files and existing files.

Note: Any subsequent [CANCEL](#) command issued after a **SAVE** has been done will only discard changes made since the last save, **not** all changes made since the beginning of the entire edit session.

File format issues

If you wish to save a file in a format other than the normal default specified by your PROFILE settings, see ["Handling Non-Windows Text Files"](#) for additional information.

SAVEALL - Save All Current Tabs

Syntax

SAVEALL	[COND]
----------------	-----------------

Operands

COND **COND** causes **SAVEALL** to only save edit files which are currently in a modified state. Edit files which are not currently modified will not be saved.

Description

SAVEALL will save all files that are currently opened for Edit. The **SAVEALL** command is equivalent to issuing the **SAVE** command on every edit file tab at the same time.

SAVEALL by itself will unconditionally save all edit files, whether currently in a modified state or not (just as a **SAVE** command will unconditionally save the one file you are editing when you issue it). **SAVEALL COND** will save all edit files that are in a **modified** state, but **unmodified** edit files will **not** be saved.

SAVEALL ignores any files that are open for Browse mode.

SAVEALL will save multiple files opened in a Multi Edit session, the same as if **SAVE** were issued in the file tab of the Multi Edit session.

If you wish to utilize any other types of file-saving actions, such as **REPLACE**, **SAVEAS** or **CREATE**, or if you wish to save a file in a format other than the normal default specified by your **PROFILE CRLF** setting, these actions must be taken individually for each desired file, because there are no "ALL" versions of these other commands.

Note: Any subsequent **CANCEL** command issued after a **SAVEALL** has been done will only discard changes made since the last save, **not** all changes since the beginning of the entire edit session.

SAVEAS - Save as New Name and Switch to it

Syntax

SAVEAS	[file-name]
--------	---------------

Operands

file-name The name of the file you wish to create. In a multi-edit session (see below) you cannot specify a file-name.

Description

SAVEAS will use all of the Edit data to create a new external file. (It is not necessary, and not allowed, to specify **.ZFIRST .ZLAST** to get every line saved.)

If you specify a simple (unqualified) file-name, then the file will be created in the same directory as the file being edited. A relative file-path (such as **MYPATH\myfile.txt**) will be created relative to the directory where the file is being edited. If the file-name is a fully-qualified name, it is created directly where you specify.

If you do not specify a file name, a conventional Windows save dialog will be presented to allow you to select the location and file name to be created.

Note 1: If the filename already exists, you will not be allowed to **SAVEAS** using that name; you will see the message, **File exists, use REPLACE to re-use it**. Like the message says, in that case, use the [REPLACE](#) command instead.

Following creation of the new file, the current Edit Tab will be switched to the newly created file, replacing the tab for the original file which will no longer be displayed. This is identical to the way 'Save As' is handled by conventional Windows editors.

Note 2: If the current edit data has not been saved in the original file and if you wish any changes to be saved there, do so **before** issuing the **SAVEAS** command. Otherwise, the changes will only be saved in the new file created by **SAVEAS**; the changes will not be saved in your original file, and you will receive no notice that this has occurred.

Operation of SAVEAS in a Multi-Edit Session

You can issue a **SAVEAS** command in a multi-edit session as normal; but no file-name operand is accepted.

This will bring up a directory-browse dialog, where you choose a directory (or create one) where all the files in your multi-edit session are saved. Once you do this, you will begin a new multi-edit session in which each **=FILE>** line will have the same file name and the (new) directory you have chosen.

You can do the **SAVEAS** selecting an existing, non-empty directory, but if any existing files have the same basic file names as the ones you are trying to save, the **SAVEAS** operation will be canceled. (**SAVEAS** will not write over existing files.) Thus, the normal and preferred procedure is to create a new empty directory (or take steps to ensure that the **SAVEAS** directory is empty ahead of time) as part of your **SAVEAS** process.

When you begin a multi-edit session using files originating from more than one directory, SPFLite permits you to have files in different directories with the same basic file name in the same session. However, if you intend to do a **SAVEAS** command, all of the basic file names involved must be unique, since the **SAVEAS** command will save

all of the files from your multi-edit session into the same directory. If they are not unique, you can continue to use your multi-edit session, but you will not be able to issue a **SAVEAS** command.

When you do a **SAVEAS** command, every file currently in the multi-edit session is saved to the new location, even files not currently in a modified state.

SC - Sentence-Case a Range of Lines

Syntax

```
SC          [ line-control-range ]
           [ X | NX | ALL]
           [ U | NU ]
           [ MX | DX ]
```

Operands

line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
MX	MX requests that all lines which are processed be excluded from the display following command processing. MX = Make excluded.
DX	DX requests that lines which are processed, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change exclusion status.

Description

The lines processed will be converted to sentence-case. Sentence case will upper-case the first letter of the first word in each sentence, and all remaining letters in the sentence will be converted to lower-case.

Scroll Commands - UP / DOWN / LEFT / RIGHT

Syntax

UP		[number	
DOWN			PAGE	
LEFT			HALF	
RIGHT			DATA	
			MAX	
			CSR]

Operands

number	The number of lines or columns to scroll; may be one to four digits.
PAGE	Scroll by the number of text lines on the screen (vertical) or the number of columns (horizontal). If the current file Profile is set to EOLAUTO or EOLAUTONL, then the scroll will be to the relative =PAGE> line of the previous/next page.
FULL	Same as PAGE except it will always ignore the EOL AUTO and EOL AUTONL settings.
HALF	Scroll the screen one half the screen width/height.
DATA	Scroll the screen one screen width/height minus 1 line/column.
MAX	Scroll the screen all the way to the top/bottom/left/right of the data depending on the scroll direction.
CSR	Scroll the screen so the line/column where the cursor is currently located becomes the relative left/right/top/bottom of the screen depending on the scroll direction. If the cursor is not currently located within the text area, this is treated as PAGE .

Abbreviations and Aliases

PAGE can also be spelled as **P**

FULL can also be spelled as **F**

HALF can also be spelled as **H**

DATA can also be spelled as **D**

MAX can also be spelled as **M**

Description

The four scroll commands are used to reposition the screen display to a different portion of the data being edited. Normally these commands are assigned as defaults of one or more keyboard keys. The **UP/DOWN** keys may not need to be further tailored by the user in KEYMAP, because they are assigned default mappings to the PageUp and PageDn keys at installation time.

Since standard command key processing always prefixes the command line with whatever command key is pressed, the operands are normally entered on the command line and then the command Key is pressed.

For example, to scroll down to the bottom of the data being edited:

- Enter **MAX** or **M** on the command line
- Press the key for **DOWN** (usually the PageDn key)
- The resulting command issued internally would become **DOWN MAX** and the screen would be positioned at the bottom of the data

If no operand is entered, the command will use the value from the **Scroll** field in the upper right corner of the screen. The value of this field may be changed at any time to any valid scroll operand and it will remain the default until you change it again.

When scrolling commands are mapped to keys, it is useful to have them prefixed with ! exclamation. That will cause any existing command in the primary command area to be cleared out first. Doing that will help prevent the scroll command from being "combined" with anything left over in the command line, which would result in an SPFLite command syntax error.

Because scrolling to the top and bottom of a file is frequently needed, it is useful to have **UP MAX** and **DOWN MAX** mapped to keys. A helpful mapping may be to have either **Alt PageUp** or **Ctrl PageUp** mapped to **!UP MAX**, and **Alt PageDn** or **Ctrl PageDn** mapped to **!DOWN MAX**.

SET - Set a Command Variable

Contents of Article

[Syntax](#)
[Operands](#)
[Description](#)
[Sample SET Edit session](#)
[Using SET to Define Command Aliases](#)

Syntax

```
SET      [ name [ OFF | { = value } ] ]
SET      name PUSH [ = value ]
SET      name POP
```

Operands

name Defines the name of the **SET** variable. A **SET** variable name is case-insensitive, and must be a letter followed by zero or more letters, digits and underscores.

SET variable names may also contain dot characters. Dots may appear anywhere in the name, except for the first position which must be a letter. Names may contain consecutive dots, may end in one or more dots, and dots may be followed by any of the legal characters: letters, digits and/or underscores.

Because the command notation **=X** is a shortcut for the command **EXIT**, a **SET** variable with a name of **X** is not allowed; the name **X** is reserved. (This shortcut emulates a notation used in IBM ISPF to quickly exit that editor.)

SET name by itself will display the current contents of **SET** variable **name**.

SET with no operands will bring up a **SET** Edit session.

OFF **SET name OFF** will remove the **SET** variable **name**. Once removed, attempting to remove it again will produce the message **Unknown SET variable**.

If the **SET** variable **name** has been *pushed*, and has one or more 'prior' values in addition to its current value, **all** of the values are deleted by **OFF**.

= value **SET name = value** will assign **value** to **SET** variable **name**. If the **value** contains embedded blanks, it must be quoted.

If the **value** contains quotes, you must enclose the entire value in other quotes. For example, suppose you want to reference a number of 12345 as a string by using the **SET** variable name **NUM** in several **CHANGE** commands, such as:

```
CHANGE ABC =NUM
```

If you defined this as **SET NUM = '12345'** this would not work, since the value would just stored as 12345 without the quotes, and the **CHANGE** command would

effectively be

CHANGE ABC 12345

and the 12345 would look like a line number, not a string. To store the **SET** variable's value so that the quotes are included, you would have to "quote the quotes". The correct command would be **SET NUM = "'12345'"**.

Then the **CHANGE** command would get correctly expanded to

CHANGE ABC '12345'

NOTE: The operands of **SET** must be separated by blanks. That is,

SET LINE = 123

is valid, but

SET LINE=123

is invalid syntax, and will result in **Unknown SET variable: =123**. This happens because SPFLite tries to expand the '=123' part of the command as if it were a SET variable.

You can use this fact to define 'symbolic' SET variables if you wished. Example:

```
SET N = 10
SET LINE=N = TEN
```

The first line sets **SET** variable N to 10. The second line is first parsed, and since N is defined, **LINE=N** is converted to LINE10. The effective command is, **SET LINE10 = TEN**, and the string "TEN" is assigned to the **SET** variable LINE10.

PUSH

The **PUSH** keyword will cause the current value of the **SET** symbol 'name' to be pushed on a 'stack', so that the current value of the variable is saved (and could be restored later with **POP**). If you specify a **value** on the **SET** command, like **SET name PUSH = value**, that will be the new value of the **SET** symbol. The 'stack' mechanism treats each name separately, so pushing and popping one **SET** variable has no effect on any other **SET** variable.

If you do not specify a value, and just say **SET name PUSH**, a duplicate copy of the current value will be placed on the stack. When this is done, the *name* used must be a currently defined **SET** symbol, otherwise you will get the error message, "Unknown **SET** variable".

POP

The **POP** keyword will cause the current value of the **SET** variable 'name' to be discarded when **SET name POP** is issued, and the prior value of that variable that was stored on the 'stack' is restored.

The **name** used must be a currently defined **SET** symbol, otherwise you will get the error message, "Unknown **SET** variable".

The **name** used must also have at least one stacked value present for **POP** to be performed. That is, in addition to setting the original value of the variable, you have to have issued a **SET name PUSH** command at least once. If this was not done, you will get the error message, "No additional values to POP for: *name*". That means you cannot "POP a **SET** name out of existence" by issuing the **SET name POP** command too many times. To

get rid of a **SET** variable, the command **SET name OFF** is required, or else you can enter a SET Edit session and delete the definition manually.

Description

The **SET** command is used to store named values, known as **SET** variables. A **SET** variable is comparable to a Windows environment variable in the way it is defined and used.

In SPFLite primary commands, you can access the value of a **SET** variable by putting an = equal sign followed by the name of the variable. Suppose you wanted to refer to a certain line number by name. You could say,

```
SET LINE = 1234
```

and then use it in commands as a symbolic name for 1234:

```
CHANGE ABC DEF =LINE
```

or

```
LOC =LINE
```

Wherever you said =**LINE** the value of LINE, 1234, would get substituted.

The substitution happens right away, so if you **RETRIEVE** a command that had a **SET** variable, the resolved value is what gets retrieved, not the =name notation. So, in the examples above, what gets retrieved will be

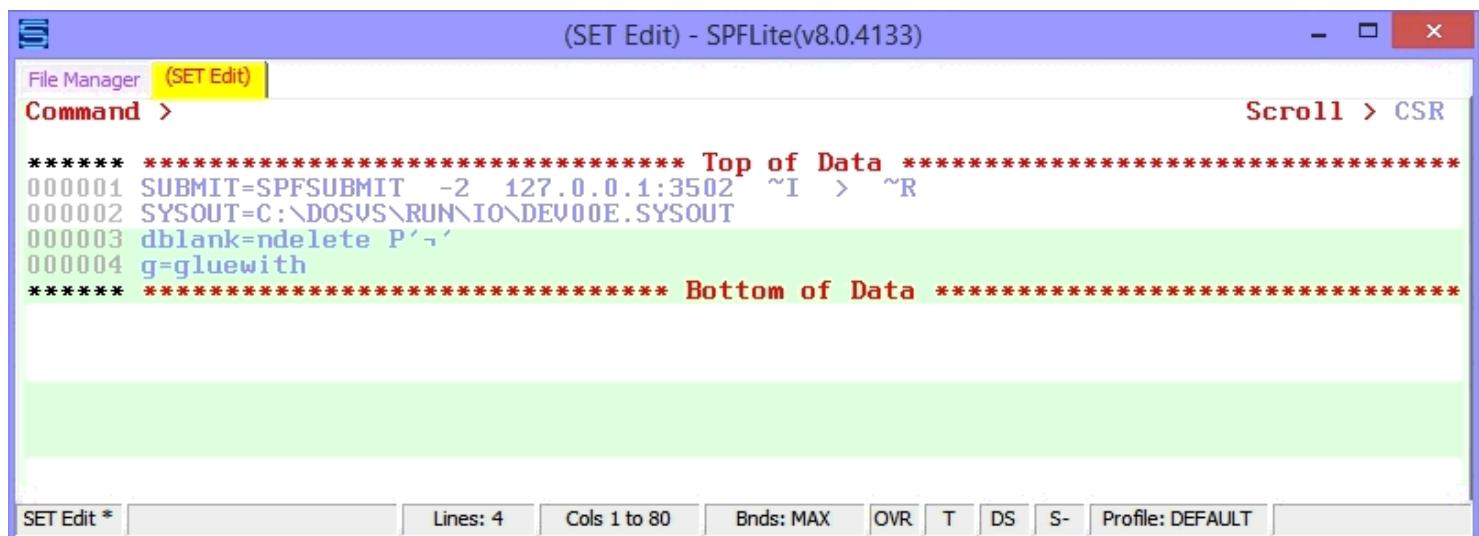
```
CHANGE ABC DEF 1234
```

and

```
LOC 1234
```

Sample SET Edit session

If you issue the **SET** command by itself, you will bring up a **SET** edit session. This is marked by a file tab of (**SET Edit**).



```
***** **** Top of Data ****
000001 SUBMIT=SPFSUBMIT -2 127.0.0.1:3502 ~I > ~R
000002 SYSOUT=C:\DOSVS\RUNNIO\DEV00E.SYSOUT
000003 dblank=ndelete P'`'
000004 g=gluewith
***** **** Bottom of Data ****
```

SET Edit * Lines: 4 Cols 1 to 80 Bnds: MAX OVR T DS S- Profile: DEFAULT

To ensure the integrity of the **SET** values, the **SET** command cannot be issued from *any* tab when a **SET** Edit session is active.

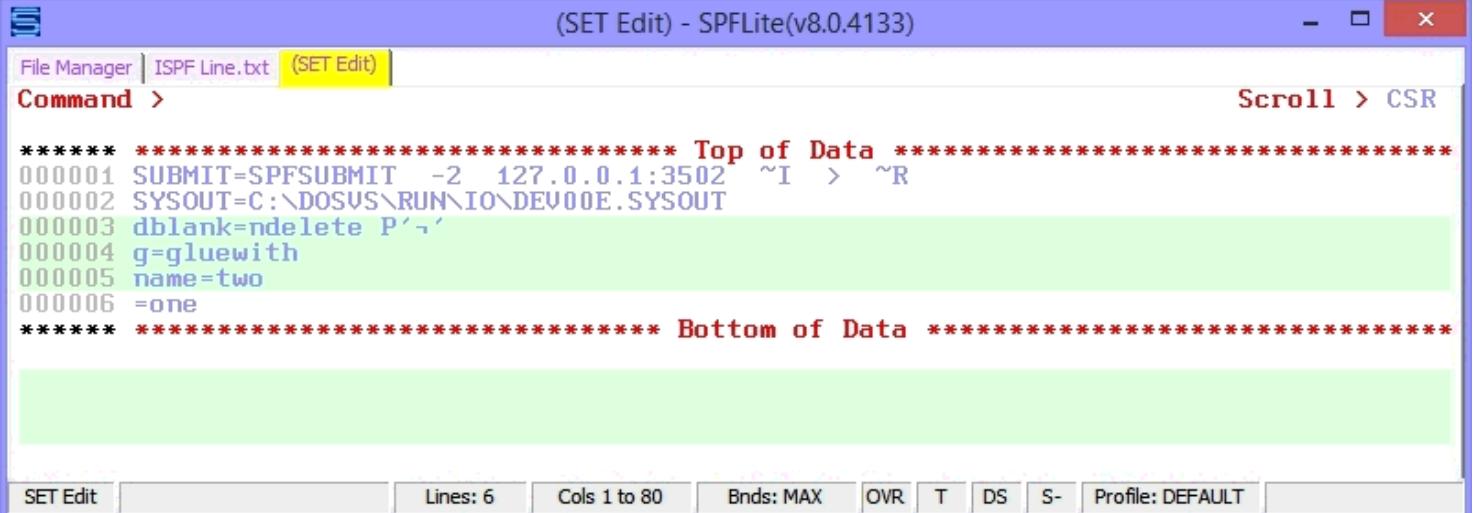
You can add, change or delete **SET** variables in the **SET** Edit session, or just browse their contents. When you issue an **END** command (usually mapped to F3), the contents of the **SET** Edit session become the new values

of all SET variables, and can be used immediately. It's not necessary to say **SAVE** in a SET Edit session. That is implied, and SPFLite takes care of that for you. However, it is possible to say **CANCEL** in a SET Edit session, which will discard any changes you might have made.

The **SET** variable values are stored in a file called **SPFLite.SPS** which is normally in the SPFLite directory under My Documents. As you will see if you start a SET Edit session, the values are stored in a straight-forward manner, as *name=value* entries, one per line, in the same manner that Windows environment variables would be stored if you issued the DOS command **SET > MYFILE .TXT** and then opened that file for editing. This fact makes it easy to import environment-like values from outside SPFLite and use them as SET variables.

Note: As you observe the SET Edit display above, the name and value are separated by a single = sign with no spaces between. This is not the same format as the **SET** command, which requires spaces between the operands. **Yes, we know it isn't consistent** - it's just the way it is.

When you **PUSH** a value on a SET variable, and then issue the SET command with no arguments, you can see how the 'stack' is stored for that name. Let's say you issued **SET name = one**, and then **SET name PUSH = two**. (The keywords are capitalized just to emphasize them; you can type them in upper or lower case, as usual.) After this was done, issue a SET with no arguments. Here is what it will look like:



The screenshot shows the SPFLite SET Edit window with the title bar '(SET Edit) - SPFLite(v8.0.4133)'. The menu bar includes 'File Manager' and 'ISPF Line.txt' (which is highlighted in yellow), and 'SET Edit'. The command line shows 'Command >'. The main area displays the variable stack:

```
***** * ***** Top of Data *****  
000001 SUBMIT=SPFSUBMIT -2 127.0.0.1:3502 ~I > ~R  
000002 SYSOUT=C:\DOSVS\RUN\IO\DEV00E.SYSOUT  
000003 dblank=ndelete P'`'  
000004 g=gluewith  
000005 name=two  
000006 =one  
***** * ***** Bottom of Data *****
```

The status bar at the bottom shows 'SET Edit', 'Lines: 6', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'DS', 'S-', and 'Profile: DEFAULT'.

Here, you can see that the current "top of the stack" for *name* is "two", and the definition of the variable looks like all the other ones, stored as VariableName=Value. But the 'old' value appears underneath, with just a leading = sign. When you **POP** a variable, this extra line is removed and only the 'current' value remains.

SPFLite takes these actions automatically and you don't normally need to be concerned with it, unless you are going to edit the values in the SET Edit session, or import these values from outside SPFLite. If you were going to do that, the main thing to remember is not to **SORT** the data in the SET Edit session, if there are any pushed values. Otherwise, the 'top' value and the 'previous' value(s) could get separated, and any subsequent **POP** commands wouldn't work right.

Using SET to Define Command Aliases

You can define command aliases for primary commands using the SET command.

A command alias name applies to both an Edit/Browse session and to the File Manager.

You cannot define an alias to override an existing command. For example, an alias cannot change BROWSE into EDIT. Aliases come into play after SPFLite determines that a given primary command is not a built-in command name, and before it looks for a possible macro definition.

You define a primary command alias by the following **SET** command:

```
SET ALIAS.name = command
```

The **name** portion of the SET variable specifies the new alias name you want to define. It should not be any existing built-in SPFLite command.

There is nothing to stop you from doing that, but you do so, SPFLite will never use your definition. For example, if you tried to say SET ALIAS.BROWSE = EDIT and then typed BROWSE as a primary command, SPFLite will never look at your ALIAS definition, and it will not launch an EDIT command.

The **command** portion of the SET command specifies the primary command to be executed when you type **name** on the primary command line. **command** may be the name of a macro or an SPFLite built-in primary command.

command may optionally contain operands. When you type **name** on the command line and follow **name** by operands, those operands will appear after the **command** you defined, including any operands of its own that may be there.

Aliases are normal SET names, and all rules for SET names apply to SET aliases. To delete an alias, you can issue **SET ALIAS.name OFF**, or enter a SET Edit session and delete the line containing the SET alias definition.

NOTE: Alias names are only recognized as such as the **first** word on the command line. If an alias name appears anywhere else, it is an ordinary SPFLite command operand with no special meaning.

How does this differ from defined SET names available previously?

Prior versions of SPFLite allowed general-purpose names to be defined. When you use them on the command line, it is necessary to put an = equal sign in front of them. So, if you wanted E to mean "EDIT", you would issue

```
SET E = EDIT
```

and then to use this E value as a command to edit file ABC.TXT, you have to put this on the command line:

```
=E ABC.TXT
```

With SET aliases, the = equal sign is no longer needed, as the following example shows:

Example

To define E as an alias for the primary command EDIT:

```
SET ALIAS.E = EDIT
```

and then to use this E value as a command to edit file ABC.TXT, you put this on the command line:

```
E ABC.TXT
```

SETUNDO - Control Undo Levels

Syntax

SETUNDO	[n]
----------------	--------------

Operands

n The number of **UNDO** levels. A maximum of 25 levels are supported. If you specify **n** as **0** it disabled **UNDO** support

Description

SETUNDO controls whether SPFLite will support the **UNDO** command, and if enabled, the number of rollback levels of **UNDO** that are supported.

SETUNDO with no operands will report the current **SETUNDO** status, which is the number of **UNDO** levels currently in effect, or 0 if disabled.

The value is stored as part of the PROFILE, which is maintained individually by file type.

When the **n** value is zero, **SETUNDO** is effectively OFF; otherwise it is ON. **SETUNDO** does not actually take the keywords **ON** or **OFF**.

SORT - Sort the Edit Data

Syntax

```
SORT      [ criteria-1 ] ... [ criteria-5 ]
          [ line-control-range ]
          [ UNIQ | MARKUNIQ ]
          [ X | NX ]
          [ U | NU ]
          [ MX | DX ]
```

Operands

criteria-1
criteria-2

You may specify up to **five** sort fields. Each criteria is of the following format:

start-col [end-col] [direction [case]]

start-col The left column of the sort key. **If not specified, 1 is assumed.**

end-col The right column of the sort key. **If no end-col is specified, the longest line length is used.**

direction A for ascending sort; D for descending sort.

case C for a case-sensitive sort; T for case-insensitive sort.

See the information on Stable sorting below regarding the automatic addition of a sequence number to each sort key.

line-control-range

The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#). Refer to that section of the documentation for details.

UNIQ

UNIQ requests **SORT** to drop duplicate records from a group of lines having the same sort key(s). If no sort keys are supplied, **SORT UNIQ** will use the assumed defaults of 1 and max line length and will remove duplicate lines based on that assumed key. That is, in the absence of **criteria** options, **SORT** will use a key equal to the maximum line length (padding with spaces if needed) to determine if records are unique or not. **SORT** will report on the number of duplicates dropped during sorting.

Because a **UNIQ** sort could result in fewer records returned than originally provided, if the original records being sorted were in non-contiguous areas, such as spans of excluded lines or lines having a given line tag, some of those areas could end up with fewer records or no records at all. Depending on the distribution of such non-contiguous lines and the values of the data lines, the exact distribution of data across such non-contiguous lines after duplicates are removed may or may not be repeatable. You should check your **SORT UNIQ** output carefully when using it in a non-contiguous data environment to be sure you have the desired results.

Note: The processing done by **SORT UNIQ** in removing duplicate records is **different**

than that used by **DELETE DUP**. See [DELETE - Delete Selected Lines](#) for more information.

Note: The command **SORT UNIQ** will delete all non-unique lines. If you want to delete all unique lines, leaving only lines having duplicate keys, first use **SORT MARKUNIQ** with the desired sort-key operands, and then issue the primary command **DELETE ALL U**.

MARKUNIQ

The **MARKUNIQ** keyword will cause all lines having unique sort keys to be flagged as User lines (**U** lines). When you use **SORT MARKUNIQ**, no lines are deleted the way that **SORT UNIQ** can do. Recall that when a line is marked as a User line, it is flagged with a **|** vertical line in the "gap column" between the sequence number and column 1 of your data. See [Working with User lines](#) for more information.

If you wanted the non-unique lines (lines with duplicate sort keys) marked instead of the unique ones, you can use the [TU](#) line command to toggle the User state of your file after issuing **SORT MARKUNIQ**. The block line command [TUU](#) can be used to reverse the User status of a block of lines. If you want the User status of the entire file to be reversed, you can place a line command of [TU/](#) on line 1 of your file and press Enter. Keep in mind that the action performed by [TU](#) and [TUU](#) is repeatable, so that if you decide later you want the User flags back the way they were, just issue the same [TU](#) or [TUU](#) command a second time for the same line range you did at first.

X | NX

Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be examined, **NX** requests only non-excluded lines are to be examined. If neither **X** or **NX** are specified, all lines in the range will be examined.

U | NU

Specifies a subset of the line range to be processed. **U** requests only User lines are to be processed, **NU** requests only non-User lines are to be processed. If neither **U** or **NU** are specified, all lines in the range will be processed.

MX

MX requests that all lines which are processed be excluded from the display following command processing.
MX = make excluded.

DX

DX requests that lines which are processed and which, if excluded, would normally be made visible, be left in their excluded status. **DX** = Don't change exclusion status.

Description

For **SORT** with no operands, the editor will default to an ascending sort on columns 1 through the maximum line length (padding with blanks if needed); if only a start-col operand is provided for one of the criteria-n, then the key will be from the specified column through the maximum line length.

The **A | D** operand may precede or follow the column operands, or may be omitted if an ascending sort (the default) is desired.

If the Case modifier is not present on the **A/D** operand, then the **SORT** will use the default set by the [CASE](#) option.

Examples

SORT 1 10 A

This will sort the data in ascending order of columns 1 thru 10.

SORT 22

This will sort the data in ascending order of columns 22 thru max line length.

SORT 5 12 DC 21 30 AT

This will sort the data in descending, case sensitive order on columns 5 through 12, secondary sort on columns 21 through 30 in ascending case insensitive order.

SORT 2 22 A :ABC

This will sort the data in ascending order on columns 2 through 22. The line-range-operand specifies **:ABC** which indicates only lines tagged with the tag **:ABC** are to be sorted. These lines could be scattered throughout the file, they will be sorted without disturbing the order of the remaining non-tagged lines in the file.

Difference from IBM's ISPF SORT command

Unlike ISPF, SPFLite's **SORT** implicitly changes excluded lines to unexcluded during the course of a **SORT X** command. To achieve the effect of an ISPF command **SORT X**, it is necessary to specify this as **SORT X DX** instead.

The SORT Command performs stable sorting of data

The **SORT** primary command performs a **stable** sorting operation; this maintains the original ordering of data lines when the sort keys are considered equal. This is achieved by SPFLite internally adding the sequence number of each line as a hidden, low-order sort key on every line. This happens automatically and no user action is required.

Application Note: Performing a Dictionary Sort

As a consequence of stable sorting, it is now possible to perform a “dictionary” sort, in which words which are all-caps appear first, then words with an initial cap, and finally words in lower case appear last, where the same underlying word is involved. A dictionary sort is done by sorting a line range first in case-sensitive mode, then in case-insensitive mode. This in turn is done by specifying **SORT C** and then **SORT T**, or by issuing a **CASE C**, then **SORT**, then **CASE T**, then **SORT** again.

Example

Original lines:

```
000001 BBB
000002 aaa
000003 Bbb
000004 Aaa
000005 bbb
000006 AAA
```

After **SORT C**:

```
000001 AAA
000002 Aaa
000003 BBB
000004 Bbb
000005 aaa
000006 bbb
```

After **SORT T**:

000001	AAA
000002	Aaa
000003	aaa
000004	BBB
000005	Bbb
000006	bbb

Now, the data is in dictionary-sorted order.

SHOW - Show Lines Where a String is Found

Syntax

```
SHOW [ search-string ]
      [ start-column [ end-column ] ]
      [ FIRST | LAST | NEXT | PREV | ALL ]
      [ PREFIX | SUFFIX | WORD | CHAR ]
      [ line-control-range ]
      [ U | NU ]
      [ color-selection-criteria ]
      [ TOP ]
```

Operands

search-string	The search string that identifies the lines to be removed from exclude status.
start-column	
end-column	Right column of a range (with start-column) within which the search-string value must be found.
FIRST	Starts at the top of the data and searches ahead to find the first occurrence of search-string.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of search-string.
<u>NEXT</u>	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
PREFIX	Locates search-string at the beginning of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
SUFFIX	Locates search-string at the end of a word.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.

U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

You can use the **SHOW** command to find a search string, and unexclude the lines that contains the string from the display. Note that the normal selection options of **X** and **NX** are not allowed, since **SHOW** operates **only** on excluded lines. You may also wish to review "["Working with Excluded Lines"](#)" for more information.

To unexclude the next excluded line that contains the letters **ELSE** without specifying any other qualifications:

On the Command line, type:

SHOW ELSE

Press Enter. Since no other qualifications were specified, the letters **ELSE** can be:

- Uppercase or a mixture of uppercase and lowercase.
- At the beginning of a word (prefix), the end of a word (suffix), or the entire word (word)
- Anywhere within the current boundaries.

To unexclude the next line that contains the letters **ELSE**, but only if the letters are uppercase:

On the Command line, type:

EXCLUDE C"ELSE"

and press Enter.

This type of exclusion is called a character string exclusion (note the **C** that precedes the search string) because it excludes the next line that contains the letters **ELSE** only if the letters are found in uppercase. However, since no other qualifications were specified, the exclusion occurs no matter where the letters are found on a non-excluded line, as outlined in the previous list

SOURCE - Specify Character Encoding

Syntax

SOURCE	{ encoding-format }
---------------	----------------------------

Operands

encoding-format	ANSI UTF8 UTF16LE UTF16BE EBCDIC user-format
------------------------	---

Abbreviations and Aliases

ANSI can also be spelled as **ASCII**

UTF16LE can also be spelled as **UTF16**

Description

The **SOURCE** command is used to change the **SOURCE** attribute of a Profile. It defines the character-set encoding format of all files with a given file type.

The **encoding-format** of a file is both its character set, and the way in which the character set is encoded as bits. For **ANSI** and **EBCDIC**, these two concepts essentially mean the same thing. For Unicode files, there is more than one way to encode them, and SPFLite supports the encoding forms noted above. (There are additional ways to encode Unicode, but SPFLite only supports the ones listed.)

The current encoding format is displayed in the status bar at the bottom right of the screen.

The **SOURCE** value is stored as part of the PROFILE options which are maintained individually by file type.

Limitations of SOURCE Encoding

At present, regardless of the **SOURCE** encoding of a file type, all files are internally processed using the **ANSI** character set while being edited or browsed. This means that only 256 possible characters are representable in SPFLite, and they must be within the Ansi character set. For the predefined **EBCDIC** to **ANSI** translation table, this is not a problem, because the translation used by SPFLite is lossless.

For Unicode it is another matter. Unicode can represent far more characters than **ANSI** can represent. To successfully use SPFLite to edit Unicode-encoded files, the character set of such files must be limited to those characters that have a counterpart in the Ansi character set. Sometimes this may be enough for editing UTF-8 encoded files such as HTML web pages, which are often merely UTF-8 conversions of simple ANSI files.

If you have extensive Unicode requirements (like editing files in Greek or Russian), these may be best met by a full-blown word processor like Microsoft Word rather than SPFLite.

EBCDIC Considerations

For **EBCDIC** files, the **SOURCE** format must be established in the file profile **before** the file is loaded since SPFLite needs to use the **SOURCE** value in the Profile while loading the file. It needs this information to know ahead of time that it needs to make use of an **EBCDIC** to **ANSI** translation table.

The first time you access an **EBCDIC** file type not referenced before is problematic, since since there would be no current existing Profile for it. The easiest way to resolve this is to do a **PROFILE NEW** command to create a

new Profile for that file type, and set its **SOURCE** attribute to **EBCDIC**, before you first edit a file that is **SOURCE EBCDIC**. See the [PROFILE](#) command for more information.

Working with user-defined translation tables

The **EBCDIC** translation table name is no longer a reserved, predefined keyword. Instead, it exists as a file with the name of **EBCDIC.SOURCE** in the SPFLite directory. This file is installed by default with the rest of SPFLite. You are free to create your own translation tables. For example, if you had your own variation of EBCDIC, you could call the file **MYEBCDIC.SOURCE**, and then enable it within your profile by issuing the command, **SOURCE MYEBCDIC**.

You are free to name your own translation-table files what you wish, except that the (non EBCDIC) names listed above are reserved.

You can replace the installation-defined **EBCDIC.SOURCE** file if you really want to, but it is recommended that you not do that, so that the default EBCDIC table would always be available for you if you ever needed it. For example, suppose you want to translate between IBM code page 1047 and **ANSI**. You could call such a table **CP1047.SOURCE** and enable it with **SOURCE CP1047**.

The **.SOURCE** translation files are now in a new format from prior SPFLite releases.

See [Handling Non-Windows Text Files](#) for more information.

Unicode Considerations

- SPFLite will normally process Unicode files automatically, and write the edited results back in the same format as the original file. The **SOURCE** command allows you to alter the desired output format during the edit and before a file is saved. Note that if you change the **SOURCE** attribute of a Profile, **every** file of that Profile's type will have that **SOURCE** attribute, not just the one you are currently working on.
- **UTF16BE** means 16-bit UTF in Big Endian format, the standard format used on mainframe systems.
- **UTF16LE** means 16-bit UTF in Little Endian format, the standard format used in Windows.
- Both **UTF16** and **UTF16LE** mean the same thing; when either is chosen, **UTF16** will be displayed in the status bar.

SPLIT - Split Lines Using Find/Change Strings

Contents of Article

[Syntax](#)

[Operands](#)

[Description](#)

[SPLIT and virtual highlighting pens](#)

[Splitting lines with labels and tags](#)

[Splitting strings vs. splitting lines](#)

[Performing splits more complex than SPLIT can support](#)

[Splitting zero-length lines](#)

[Line splitting and line exclusion](#)

[Using ALL FIRST and ALL LAST on SPLIT](#)

[Using ALL FIRST and ALL LAST elsewhere](#)

[A note about the IBM ISPF legacy SPLIT command](#)

Syntax

```

SPLIT      from-string
          { P'to-string' | F'to-string' }
          [ start-column [ end-column ] ]
          [ FIRST | LAST | NEXT | PREV | ALL ]
          [ PREFIX | SUFFIX | WORD | CHAR ]
          [ LEFT | RIGHT ]
          [ line-control-range ]
          [ color-selection-criteria ]
          [ X | NX ]
          [ U | NU ]
          [ TOP ]

```

Operands

from-string The search string you use to describe the text to be split. This text, when found, is discarded and replaced with the to-string, described next.

P'to-string' | **F'to-string'** The string you want to replace from-string, which **must** be defined as a P-type Picture string or an F-type Format string, which contains one, and only one, vertical bar | character to define the *split-point*.

start-column Left column of a range (with end-column) within which the from-string value must be found. If no end-column operand, then the from-string operand must be found starting in start-col.

end-column Right column of a range (with start-column) within which the from-string value must be found.

FIRST Starts at the top of the data and searches ahead to find the first occurrence of from-string. See the discussion below about using **ALL FIRST** and **ALL LAST** on **SPLIT**.

LAST Starts at the bottom of the data and searches backward to find the last occurrence of

from-string. See the discussion below about using **ALL FIRST** and **ALL LAST** on **SPLIT**.

NEXT

Starts at the first position after the current cursor location and searches ahead to find the next occurrence of from-string. **NEXT** is the default.

PREV

Starts at the current cursor location and searches backward to find the previous occurrence of from-string.

ALL

Starts at the top of the data and searches ahead to find all occurrences of from-string. See the discussion below about using **ALL FIRST** and **ALL LAST** on **SPLIT**.

LEFT

LEFT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the left-most occurrence of search-string is changed, and any other instances on that same line are unchanged.

RIGHT

RIGHT causes the search-string to be found at most once in any given line. Where the search-string occurs more than once in the same line, only the right-most occurrence of search-string is changed, and any other instances on that same line are unchanged.

PREFIX

Locates from-string at the beginning of a word.

WORD

Locates from-string when it is delimited on both sides by blanks or other non-alphanumeric characters.

CHAR

Locates search-string regardless of what precedes or follows it.

SUFFIX

Locates from-string at the end of a word.

line-control-range

The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#). Refer to that section of the documentation for details.

color-selection-criteria

A request for selection based on the highlight color of the from-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in ["Color Selection Criteria Specification"](#). Refer to that section of the documentation for details.

X | NX

Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be examined, **NX** requests only non-excluded lines are to be examined. If neither **X** or **NX** are specified, all lines in the range will be examined.

U | NU

Specifies a subset of the line range to be processed. **U** requests only User lines are to be processed, **NU** requests only non-User lines are to be processed. If neither **U** or **NU** are specified, all lines in the range will be processed.

TOP

Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If **TOP** is coded, then the line is always positioned as the **top** line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

The **SPLIT** edit primary command is used to selectively split apart lines of text based on a search string. After a split occurs, a line on which the **from-string** is found will become two lines. Everything before the "split point" will be on the first line, while everything after the split point will be on the second line.

Notes:

- The **JOIN** command performs the opposite function, combining two lines into one. See [JOIN - Join lines Using Find/Change Strings](#) for more information.
- **SPLIT** and **JOIN**, as new SPFLite technologies, have been extensively tested, but it is possible you may run into issues using them. If you have any questions about how these commands operate, feel free to leave us your feedback on the SPFLite web site.

The **to-string** must be a P-type Picture string or F-type Format string, which must contain one and only one vertical bar | character, and which may optionally contain other characters. The vertical bar is **not** a literal | character nor any other data value, but represents a line-break to be inserted into your data. Any characters in the **to-picture** that appear **before** the vertical bar will appear at the end of the first line, and any characters in the **to-picture** that appear **after** the vertical bar will appear at the beginning of the second line, after the split takes place.

Note: While you might be tempted to think of a line-break being "inserted into your data" as being equivalent to inserting a raw **EOL** string (such as a Carriage Return / Line Feed pair) into your data line, that is not always the case. You can also split lines on fixed-length files which are defined as **EOL NONE**. The "insertion" of a line-break should be thought of in **logical** terms, not with reference to any line delimiter characters. When your file gets saved to disk, SPFLite will handle the **EOL** delimiters as needed; you don't need to be concerned about this.

Note: While the **to-string** can be a Format string, you will find that using a Picture string here should address most of your **SPLIT** requirements. Where a Format string comes in handy is when the **from-string** is a Picture, **and** you need a code of = in the **to-string** that doesn't match the corresponding position in the **from-string**.

Note: Because the **to-string** must be a Picture or Format string, there may be cases where your "replacement data" might include characters that are already defined as special-purpose Picture or Format codes. If you need such characters treated as ordinary data rather than as Picture or Format codes, you can escape those characters by preceding them with a \ backslash. See [Specifying a Picture or Format String](#) for more information.

Because the vertical bar | character in the **to-string** represents a *split point*, but is **not** itself a literal character, when you specify the to-string literally as p | , with no other characters, you are asking the **SPLIT** command to **delete** each instance of the from-string and replace it with a line break. (Conceptually, this deletes data exactly the same as a **CHANGE** command with a to-string of ,). If you don't want your found-strings completely deleted, or if you want your found-strings replaced with something else, you have to specify that in the to-string Picture - either before the vertical bar, after the vertical bar - or both places, if you wish.

SPFLite treats the **SPLIT** command as a specialized type of **CHANGE** command. Because of this, the **RFIND**, **RLOCFIND** and **RCHANGE** commands will also apply to **SPLIT**. Traditionally, **RFIND** (or **RLOCFIND**) is mapped to F5, and **RCHANGE** to F6. This means you can selectively split lines by alternating the use of the F5 and F6 keys (or, other keys if you map these commands differently).

The **from-string** may be any SPFLite string type (except for F-type Format strings), including Pictures and

Regular Expressions. If the **from-string** is a P-type Picture string, it may contain the *alignment* Picture codes [and] if desired. Be mindful that alignment Picture codes do not represent data values, and so a search Picture cannot consist solely of alignment Picture codes. (Unlike the **JOIN** command, which *requires* the from-string to contain an alignment code, the **SPLIT** command *allows* but does not *require* the use of alignment codes.)

The **to-string** Picture may contain any number of ! Picture/Format codes. Each ! Picture code represents the entire text found by the **from-string**, even when that text can vary in size or content when it is described by a P-type Picture string or R-type Regular Expression. This capability could be very useful in some **SPLIT** situations. For example, suppose you had a "label" in your data consisting of ABC plus a digit. You could change this into two lines, where the first line ends in the label and the second line begins with a duplicate copy of the label, by issuing a command of

```
SPLIT P'ABC#!' P'!!' ALL.
```

See [Working with SPLIT and JOIN Commands](#) for example usage of the **SPLIT** command.

SPLIT and virtual highlighting pens

The **SPLIT** command does not use the various "virtual highlighting pen" color name keywords, like **FIND** and **CHANGE** do. The colors that are present after **SPLIT** completes depends on how the original found string is colored.

If the found string is entirely of one color, any text inserted by **SPLIT** will have the same color that the found string had before being split. If that text had been colored by using one of the "virtual highlighting pen" keyboard functions, that color will remain intact after the split takes place, or if the text had the default (normal) color beforehand, the new text will also have the default color.

If the found string is not entirely one color, but contains text in two or more colors (where the default text display is also considered a color), all of the text inserted by **SPLIT** will have the default (standard) color. SPFLite does things this way because it would be too hard to reliably determine how and where the newly inserted text should be colored, if multiple colors had to be assigned to the inserted text.

Splitting lines with labels and tags

When you split a line containing a label, the label will remain with the **first** line produced from the splitting. This is true even when a line is split in more than one place. For example, suppose you had a line like this:

```
.A 001 A-B-C
```

If you issue the command **SPLIT '-' P' | ' ALL .A**, **line 1** will keep the label (the extra digits on a line with a tag or label don't really appear in SPFLite; we are just showing them here for explanation purposes):

```
.A 001 A
000002 B
000003 C
```

When you split a line containing a tag, the tag is propagated on to **every** line produced from the splitting. This is true even when a line is split in more than one place. For example, suppose you had a line like this:

```
:A 001 A-B-C
```

If you issue the command **SPLIT '-' P' | ' ALL :A**, **every** line will have the tag:

```
:A 001 A
:A 002 B
:A 003 C
```

Splitting strings vs. splitting lines

Just to be clear, **SPLIT** can only split a **string** in one place, but that doesn't mean it can't split a **line** in more than one place. For example, suppose you had a line like this:

```
000001 A-B-C-D
```

If you issued the command **SPLIT '-' P'|' ALL .1**, the data on line 1 would be split apart exactly as you'd hope, and you'd get the following result. A message "Split performed 3 times" would appear, corresponding to the 3 places where the '-' dash appears on the line:

```
000001 A
000002 B
000003 C
000004 D
```

What you **can't** do is issue a command like **SPLIT 'A-B-C-D' P'A|B|C|D' .1**, because that is asking the **single** string '**A-B-C-D**' to be split **three** times, and SPFLite doesn't support that. But, what if that **is** what you wanted to do? Keep reading ...

Performing splits more complex than SPLIT can support

You may encounter cases where the **SPLIT** command won't do everything you want. You might want to split a particular string into multiple lines, whereas **SPLIT** will break a string in only one place. You may have text already colored by highlighting pens and you need precise control over how the text colors are affected by splitting. You may need special features supported by **CHANGE**, such as **LEFT**, **RIGHT**, **TRUNC**, **MX**, **DX**, etc. These and other cases may require a different approach.

Keeping in mind that a **SPLIT** is a type of "change" to a line, you can perform more-complex splitting by doing this in two stages. First, use a **CHANGE** command to insert "user-defined split points" into your data, and then go back and use **SPLIT** to actually break them apart. This technique has the nice feature that after the first part, you can go and manually inspect all of the user-defined split points you just put in, and verify they are all where you want them, possibly adding and removing a few before the second part, if you have certain special cases where some extra data must be split and other split points have to be taken out. Because the **CHANGE** command, and any manual editing of your own, will have placed these user-defined split points exactly where you need them, only a simple form of **SPLIT** will be required to break the lines apart.

To do this, you might want to map some special ANSI character that you rarely use in your own data, and use that to represent your user-defined split points. If necessary, you can use the [\(Ansi\)](#) function to get any ANSI character into the clipboard, and then use it as a value for **KEYMAP**. For example, you could use the ? character for a user-defined split point.,

Suppose your task was to split some string "AB-CD-EF" into three lines on the dashes, between lines .ONE and .TWO. You can't do that directly with **SPLIT**, but (assuming that ? does not appear in your data) you could do the following instead.

```
CHANGE 'AB-CD-EF' 'AB?CD?EF' ALL .ONE .TWO
SPLIT '?' P'|' ALL .ONE .TWO
```

So, can you see **now** how to do that three-way split we said (back in the last paragraph) you couldn't do? Remember, we said that

```
SPLIT 'A-B-C-D' P'A|B|C|D' .1
```

was illegal, and it is. But you **can** do this:

```
CHANGE 'A-B-C-D' 'A?B?C?D' .1
SPLIT '? P' | ' ALL .1
```

Just that easy.

By the way, if you don't feel like using a ? character, pick anything you like that's convenient and not already in your data.

Splitting zero-length lines

SPLIT cannot be applied to zero-length lines, since there is literally nothing for **SPLIT** to find. If you were inclined to do this, it implies that you wanted to add another zero-length line next to the original zero-length line. That is, wherever there was a blank line, you'd now have two of them.

If you really want to do this, the easiest way to do it is to find all the zero-length lines, **APPEND** as many user-defined split points as you need, and then split these characters as needed. Here is an example of converting all zero-length lines into two zero-length lines each. The commands will exclude all lines that are zero-length, then place a ? character on each excluded line (and also unexcludes them in the process) and then splits each marked line into two lines, in a way that also removes the ? character.

```
RESET
NX P'=' ALL
APPEND '? ALL X
SPLIT '? P' | ' ALL
```

The [Pad to Length command PL](#) can take a / or \ modifier. Putting **PL/** on line 1 of a file will ensure that every line of the file is at least one character long.

Line splitting and line exclusion

The **SPLIT** command supports the **X** and **NX** keywords, to allow you to limit your line-range selection to only excluded (**X**), or only not-excluded lines (**NX**), if you wish. Regardless of the use of **X** or **NX** keywords, when a line is split, it is considered a "change" to the line. Any line that was excluded at the time is was split will become two lines, and both of these lines will be **unexcluded**.

The **SPLIT** command does **not** support the **MX** and **DX** keywords at this time.

Using ALL FIRST and ALL LAST on SPLIT

Because the processing performed by **SPLIT** is different than ever done in ISPF or any prior version of SPFLite, an unusual situation may occur that could affect how **SPLIT** works, perhaps in a way you would not want.

Suppose you had some lines like this ([original data](#)):

```
000001 oneTWO oneTWOsix
000002 oneTWO oneTWOsix
```

and you wanted to split the "oneTWO" strings into "one" and "TWO", but **only** the **left**-most ones. That is, you are hoping you can change this data to look like this ([desired result](#)):

```
000001 one
000002 TWO oneTWOsix
000003 one
000004 TWO oneTWOsix
```

Remembering that **SPLIT** allows a **LEFT** or **RIGHT** operand, you might be inclined to write your **SPLIT** command like this:

```
SPLIT 'oneTWO' P'one|TWO' LEFT ALL
```

This splits the left-most occurrence of 'oneTWO' on each line. That should work, right? Unfortunately, **no**.

The reason it won't work is that the **SPLIT** command will (a) effectively split the data the way you see it above as the 'desired result', but then (b) it will **keep on** splitting the data **again**, on the lines it just split.

For example, notice above that line 2 contains '**TWO oneTWOsix**' after the first split. The **SPLIT** command, after splitting line 1, moves on to "line 2".

However, **that** line 2 is not the **original** line 2, but the **newly created** line 2. On **that** line 2, there is an instance of the string '**oneTWO**', which is the beginning of the string '**oneTWOsix**' - and so **that** instance is **also** the "left-most" of its kind on **that** line. This means that it matches the **SPLIT** command's search string of '**oneTWO**', and so it also gets split. The net result is that the file will actually end up looking like this:

```
000001 one
000002 TWO one
000003 TWOsix
000004 one
000005 TWO one
000006 TWOsix
```

So even though you **asked** for only the **left**-most string of '**oneTWO**' to be split on any given line, you got them **all** split. Well, what can be done about this?

Recall we starting this discussion by saying that the processing performed by **SPLIT** is different than ever done in ISPF or any prior version of SPFLite? This is the first time for any SPF-style editor that a primary command, under the control of find and change strings, could split apart lines while in the process of scanning them. This issue has to do with the way the keyword **ALL** works.

If you think about a standard **CHANGE** command, you could change the **NEXT** string, the **PREV** string, or **ALL** strings. Now, when you say **CHANGE ALL**, do you really **care** in what order they are **ALL** changed, as long as it gets done? Normally, **no**.

However, a **SPLIT** command could find a string on a line, and split that line, and **then** "run into" the remainder of the line it just split, possibly finding **more** of the same specified search string. That's what happened in our example. This will normally only be an issue when you are splitting a string which may have more than one occurrence on a given line.

In particular, this problem will normally only happen when using the **LEFT** operand on **ALL** lines. If you did a split of the **RIGHT** string on **ALL** lines, you would not "run into" a partial line that had more instances of the string, and so the problem we are discussing wouldn't happen. That's because **SPLIT** does its **ALL** processing going forward from the beginning of the line range you are working on to the last line of it. It's also because, once a line is split, and the right-hand side of the split point becomes a new line, the determination of what constitutes the "left side" or what is the "left most" occurrence of a string, **starts over from the beginning of that new line**.

Because **SPLIT** with the **ALL** operand performs its processing in a forward direction, the possibility of "running into" a "partial line" and re-processing it may exist.

How does **ALL FIRST** and **ALL LAST** solve this? Think of these keywords like this:

- o **ALL FIRST** looks for **ALL** search strings, from the **FIRST** line in the line range to the **last** line, doing so in

a **forward** direction

- **ALL LAST** looks for **ALL** search strings, from the **LAST** line in the line range to the **first** line, doing so in a **backwards** direction

The behavior of ordinary ISPF and SPFLite commands that take the **ALL** command has always been as if **ALL FIRST** had been specified. Except, up until now, you couldn't **do** that - **ALL FIRST** was illegal syntax. Now, you **can**.

When you have repeating data on several lines, and you only want to change the **LEFT** or **RIGHT** instance of them, you need to make sure you're using the right "form" of the **SPLIT** command to make sure you are getting the results you wanted. This means, in most cases, you will want to issue your **SPLIT** commands that have **LEFT** or **RIGHT** like this:

```
SPLIT 'oldnew' P'old|new' RIGHT ALL
```

```
SPLIT 'oldnew' P'old|new' LEFT ALL LAST
```

Keep in mind that the first of these commands is really a shorthand for

```
SPLIT 'oldnew' P'old|new' RIGHT ALL FIRST
```

but you don't need to be that explicit, because the **FIRST** is understood.

Remember that, as always, the various reserved keywords can be specified in any order you wish.

Using **ALL FIRST** and **ALL LAST** elsewhere

Because much of SPFLite uses common logic to handle common features across many primary commands, you will find that the keywords **ALL FIRST** and **ALL LAST** will be accepted on other commands, such as **FIND** and **CHANGE**.

However, because the special circumstances present for **SPLIT** don't occur for **FIND** and **CHANGE**, you will find that the **ALL FIRST** and **ALL LAST** work the same way that an ordinary, garden-variety **ALL** keyword does.

Why didn't we just skip over the rest of the line after splitting, to avoid this problem?

That's a good question. The answer is, if the **SPLIT** engine did such a thing, you would only be able to make one physical split per line. That is too big a restriction to impose on you, especially when it's very likely you would want to do that very thing, and probably quite often.

For certain, **SPLIT** is powerful, and will take some study and experimentation to get the hang of it, but if you need to break apart large numbers of lines based on find/change criteria, there's nothing else like it.

A note about the IBM ISPF legacy **SPLIT** command

SPFLite does not support the IBM ISPF feature of "3270 split screen mode" and the associated legacy **SPLIT** command. With tabbed editing and the ability to open multiple instances of SPFLite, as well as the Multi-Edit feature to edit several files simultaneously in the same edit window, IBM's 3270 split screen mode is not really needed.

The SPFLite primary command **SPLIT** described here is unrelated to the IBM ISPF legacy **SPLIT** command, and merely reuses the **SPLIT** command name for a different purpose.

START - Set Initial File Position Option

Syntax

START	[FIRST LAST PRIOR LABEL NEW]
-------	--

Operands

FIRST The edit file is positioned to line 1 when opened, the same as occurs with **TOP** or **LOCATE .ZFIRST**.

LAST The edit file is positioned to the last line of the file when opened, the same as occurs with **BOTTOM** or **LOCATE .ZLAST**.

PRIOR **PRIOR** requires **STATE** to be set **ON**.

When **START PRIOR** is in effect, the edit file is positioned to the position where the file was located in the most recent edit session. If the file has not been edited before or does not (yet) have **STATE** information stored for it, the action is the same as **STATE FIRST**.

LABEL **LABEL** requires **STATE** to be set **ON**.

When **START LABEL** is in effect, the edit file is positioned to a predefined label name of **.START** if such a label exists, as if the command **LOCATE .START** had been issued after the file is first opened. If the file has not been edited before, does not (yet) have **STATE** information stored for it, or does not have a predefined label name of **.START**, the action is the same as **STATE FIRST**. The command **START LABEL** cannot be issued if **STATE OFF** is in effect. If **STATE ON** is issued, then **START LABEL**, then **STATE OFF**, the **START LABEL** setting is retained but ignored until such time as **STATE ON** is in effect. The label **.START** must be manually set by the user to any desired line location.

NEW **NEW** requires **STATE** to be set **ON**.

When **START NEW** is in effect, the edit file is positioned to a predefined label name of **.START** if such a label exists, as if the command **LOCATE .START** had been issued after the file is first opened. If the file has not been edited before, does not (yet) have **STATE** information stored for it, or does not have a predefined label name of **.START**, the action is the same as **STATE FIRST**. The command **START NEW** cannot be issued if **STATE OFF** is in effect. If **STATE ON** is issued, then **START NEW**, then **STATE OFF**, the **START NEW** setting is retained but ignored until such time as **STATE ON** is in effect. The label **.START** is automatically placed into the last line of the file when closed.

When **START** is issued with no operands, a message is displayed that shows the current setting of the **START** option.

Description

The **START** primary command is used to specify where an edit file is to be first positioned when opened. If the specific location is not important, a setting of **PRIOR** or **FIRST** may be appropriate. Other choices are detailed above.

Note: **START PRIOR**, **START LABEL** and **START NEW** require **STATE ON** to be effect **before** you issue one of these **START** commands. However, the installation **DEFAULT** profile for **STATE** is **OFF**. If you are going to use these **START** options on a regular basis, you may wish to modify the **DEFAULT** profile so that any newly created profiles have the **STATE** and **START** settings you prefer.

See [Working with File Profiles](#) for more information about the **DEFAULT** profile.

Special Processing for **START NEW**

The intent of **START NEW** is to provide a means of opening a file and locating a point in the file where "new" information has been written to the file from an external process outside of SPFLite. For example, suppose a file was being used as a **SYSOUT** spool file as written by the Hercules emulator. By setting the **START** option to **NEW**, each time "new" information is appended to the end of the **SYSOUT** file, you can (re)open the file in SPFLite, and it will automatically position the file to the last location that you knew about previously, based on the last known location of the label **.START**, which SPFLite has stored into the file for you. Then, after you have seen the file and closed it, the **.START** label will get automatically (re)positioned to the last physical line of the file, as of that time. After more data gets written to the end of the file, the process can be repeated.

By using **START NEW**, you no longer need to manually search for where the most recently-added data lines are (somewhere) near the end of the file, because SPFLite will automatically take you to the exact location you need to be in.

It is expected that many users (especially user of Hercules) will combine **STATE ON**, **START NEW**, and **EOL AUTO** or **AUTONL** to allow easy viewing of **SYSOUT** files.

Note about ISPF START

In IBM ISPF, the **START** command starts a dialog in a new logical screen. The SPFLite **START** command, as described above, performs a completely different and unrelated function.

STATE - Control Saving of Edit State Information

Syntax

```
STATE | STATS [ ON | OFF | MOST | FEW | DELETE | CREATE ]
```

Operands

ON	This indicates that you would like all files controlled by this Profile to be processed by STATE.
OFF	This indicates that no files controlled by this Profile are to be handled by STATE.
MOST	This is similar to STATE ON. If you take no specific action for a specific file, then it acts just like STATE ON, all files will be processed by STATE. However it supports allowing you can exempt specific files, see STATE DELETE below.
FEW	This is similar to STATE OFF. If you take no specific action for a specific file, then it acts just like STATE OFF, all files will not be processed by STATE. However you can exempt specific files, see STATE CREATE below.
DELETE	This command would be issued while editing a specific file and STATE MOST has been set. In fact, it will be rejected if these conditions are not met. Since STATE MOST, by default, will have created active STATE data for this file, the STATE DELETE will remove this active data and setup an indicator so that no future STATE processing for this file will occur.
CREATE	This command would be issued while editing a specific file and STATE FEW has been set. In fact, it will be rejected if these conditions are not met. Since STATE FEW, by default, will not have created any active STATE data for this file, the STATE CREATE will cause this active data to be created the next time the file is saved, and will setup an indicator so that STATE processing will continue for this file in future edits.

Note: The keyword **STATS** is provided to mimic the ISPF verb, as STATE ON/OFF is also the controlling setting as to whether File Manager displays the line count value for a file.

Description

SPFLite can optionally maintain the status of an edit session for a file so that you can save the file, exit SPFLite and at a later time restart the edit and it will resume where you left off. **STATE** saves all line labels, tags, exclude line status and current scroll position.

STATE processing supports multiple levels of state saving for a profile, from creating STATE information for all files under a Profile, through to not creating STATE information for **any** files under the profile. You choose the level that best suits the majority of the files needs.

Details of these levels, and how **STATE** operates can be reviewed in [Saving the Edit STATE](#).

The value is stored as part of the PROFILE options which are maintained individually by file type.

STATS - Turn file STATS ON/OFF

Syntax

STATS	operands
--------------	-----------------

Operands

Note: The command **STATS** is provided to mimic the ISPF verb. It is actually an ALIAS for the **STATE** command, which should be referred to for details.

SUBARG - Set Default SUBMIT Argument

Syntax

SUBARG	[string OFF]
---------------	-------------------------

Operands

string	The <i>string</i> operand is optional. It may be specified like a FIND string, which can be either plain text or a quoted string if it contains embedded blanks or special characters.
OFF	Nullifies the value

Description

The **SUBARG** command is used to set an optional argument for use by the **SUBMIT** command which is 'tied' to the Profile for a specific file type. It becomes available via the **~z** or **^z** variables and can thus be passed to the command file processing the **SUBMIT**.

To nullify the value, use the keyword **OFF** rather than "" (a zero-length string). Thus, **OFF** (in any upper or lower-case spelling) is reserved. Any other **SUBARG** value may be set, and is accepted as-is (the value is not upper-cased). Note that if you actually wish to use **OFF** and **ON** as values, that is possible, except that **OFF** will always appear in upper case, while an **ON** value, if used, appears just as you enter it on the **SUBARG** command. It is not possible to define the **~Z** code as an empty string. The value **OFF** is handled the way it is to make checking for it easier in batch scripts.

What would the **SUBARG ~Z** value be used for? The submit edit file's extension is available from the **~X** code, so that can be used to tailor a submit script based on file type. Suppose a *group* of related files had some common requirement; then, the file type alone might not be enough to make the required distinction. Or, perhaps files are being submitted that do not have a file extension. The **SUBARG** value provides a means to tailor the submit process in these special circumstances.

When **SUBARG** is issued without an operand, the current state of the **SUBARG** feature (either the *string* or **OFF**) is displayed.

See [SUBMIT - Pass Lines to an External Command File](#) and [Working with the SUBMIT Command](#) for more information.

SUBCMD - Set alternate command for SUBMIT

Syntax

SUBCMD	[string OFF]
--------	--------------------------------

Operands

string	The <i>string</i> operand specifies an alternate command name to be used, instead of the normal SUBMIT command for this file type, when SUBMIT is issued. This must be the name of another SPFLite primary command (like RUN) or of a valid user Macro name which is to replace the normal processing done by SUBMIT.
	It is possible for string to have multiple operands separated by spaces. If specified this way, the string value must be enclosed in quotes.
OFF	Disables the SUBCMD feature

Description

The **SUBCMD** command is used to specify an alternate command to be substituted in place of the normal **SUBMIT** command. This value is associated with the Profile settings for this file type.

For example, suppose you wanted to treat the **SUBMIT** of a Windows Batch file as a **RUN** command, even though **.BAT** files are supposed to be **RUN** and not submitted. In the profile for **.BAT** files, you can say **SUBCMD RUN**. Then, if **SUBMIT** is issued for a **.BAT** file (whether intentionally or not), the file will actually be **RUN** instead of being submitted. This feature will prevent erroneous SUBMIT streams from being created, which can be helpful if mainframe users get into the habit of frequently issuing **SUBMIT**.

Another use of **SUBCMD** is to launch a user-written “submit macro”. To allow such a macro to perform certain processing and then issue its own “submit” process, the command **XSUBMIT** may be used. **XSUBMIT** is similar to **SUBMIT**, except that an explicit file name is used, rather than taking the submitted file from the current edit session. A possible reason for a user-written “submit macro” might be expand user-defined “include” directives.

To disable the SUBCMD feature, the value, use **SUBCMD OFF**. When the SUBCMD feature is disabled, any SUBMIT command is processed normally without it being substituted by an alternative command. The SUBCMD feature is disabled by default.

When **SUBCMD** is issued without an operand, the current state of the SUBCMD feature (either the *string* or **OFF**) is displayed.

Useful macros for SUBCMD when disabling SUBMIT

For profiles in which you don't want **SUBMIT** to be issued at all, it is not sufficient to simply set **SUBCMD** to **OFF**. **OFF** allows a normal **SUBMIT** to proceed. Instead, you may wish to define one or both of the following macros, so that a **SUBMIT** command would either do nothing, or would issue an error message. The names you select may be anything, of course; the names are a good starting point:

```
' NOP.MACRO
HALT
```

```
' SUBERR.MACRO
HALT ("SUBMIT not supported for this file type")
```

Then, you would issue a command of **SUBCMD NOP** or **SUBCMD SUBERR**, as needed.

Disabling **SUBMIT** in this way is something you might wish to do if you issue the **SUBMIT** command very frequently, and have an (unfortunate) habit of doing so on file types that shouldn't be submitted in the first place. It is better to quickly receive an error message than wait several seconds for the **SUBMIT** handler to complete before telling you that you submitted the wrong file.

See [SUBMIT - Pass Lines to an External Command File](#) and [Working with the SUBMIT Command](#) for more information.

SUBMIT - Pass Lines to an External Command File

Syntax

```
SUBMIT      [ line-control-range ]
              [ X | NX ]
              [ U | NU ]
              [ I argument-list I ]
              [ DEBUG ]
```

Operands

line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool for specifying the range of lines to be selected, and may include line labels, line-number pseudo-labels, relative labels and line tags. The full syntax and allowable operands that make up line control ranges are discussed in "Line Control Range Specification" . Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be submitted, NX requests only non-excluded lines are to be submitted. If neither X or NX are specified, all lines in the range will be submitted without regard to their exclusion status.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
[I argument-list I]	An optional list of zero or more arguments, enclosed in actual I I bracket characters. Empty brackets will be accepted but are otherwise ignored. (Note in the syntax description, the italicized, highlighted brackets are meant to be literally specified, while the non-italicized brackets are there to show that the argument-list portion is optional.) Any arguments in this list that contain embedded blanks must be enclosed in double quotes. No other quote type will work, simply because the arguments are destined for a Windows command line, and that is the format that Windows requires for arguments having embedded blanks.
DEBUG	If specified, keeps the Windows command prompt window open after the submit program is run, as a debugging aid. You will see the command and its execution in the command prompt window, and you can type in more commands if you wish. To return to SPFLite, type the command EXIT and press Enter, or close the window by clicking on the X at the top of the window. You will see on the Windows title line of the command prompt window the words, SUBMIT Debug Window - Enter EXIT to continue .

Abbreviations and Aliases

SUBMIT can also be spelled as **SUB**

Description

The SPFLite **SUBMIT** primary command allows a program or batch file to be launched from SPFLite. Prior to executing the program or batch file, a copy of an entire edit file, or selected contents of it, are written to a temporary file. In particular, this means that a modified edit file need not be saved prior to being submitted; 'temporary' changes can be made to a file, then the file may be submitted that included those temporary changes, and then finally those changes could be canceled, if desired.

Temporary files created by SPFLite are periodically purged. The purge occurs during SPFLite startup time, when all files in the **SUBMIT** Working Directory that match a file pattern of **SUB*.*** and are two or more days old are deleted. The user is free to manually delete these files sooner, if desired.

A number of *submit codes* are supported to provide arguments to the submit command. A submit code generally consists of a ~ tilde and a single letter (case insensitive), except for the ~K and ~S codes that have an extended syntax. Note: Any submit code that represents or contains a directory path as a string will contain only that path, without any enclosing quotes, even if the path contains embedded blanks. It is the user's responsibility to enclose submit codes on the **SUBMIT** prototype command in quotes when there is a possibility that the values might have embedded blanks.

Submit codes may be specified anywhere on the **SUBMIT** prototype command as needed. It would be considered unusual for a **SUBMIT** prototype command to contain more than one instance of the same kind of submit code, but SPFLite does not prevent such usage.

A submit code can alternatively be coded with a ^ circumflex instead of a ~ tilde, such as ^R. When this is done, the given submit code essentially works the same way, but the string value that is substituted for it is converted to upper case. This may be useful when the submit prototype command launches a batch script, so that the case of certain batch script arguments can be ignored.

The defined submit codes are as follows:

~1, ^1 through ~9, ^9

A specific argument number specified in the **SUBMIT** primary command that was enclosed in brackets; the enclosing argument brackets on the **SUBMIT** primary command are removed. For example, the primary command **SUBMIT [one "two three"]** causes the ~1 code to be replaced by the value of **one** and ^1 to be replaced by the value of **ONE** wherever they appear on the **SUBMIT** prototype command. Submit codes ~9 and ^9 are "catch-all" codes that represent argument 9 and beyond, if there are ten or more argument strings supplied.

~A, ^A

All arguments specified in the **SUBMIT** primary command that were enclosed in brackets; the enclosing argument brackets on the **SUBMIT** primary command are removed. For example, the primary command **SUBMIT [one "two three"]** causes the ~A code to be replaced by the value of **one "two three"** wherever ~A appears on the **SUBMIT** prototype command.

~C, ^C

The full path name of the configuration-files directory of SPFLite. A sample value for ~C would be "C:\Documents and Settings\George\My Documents\SPFLite".

~D, ^D

The current directory date of the original file, in the ISO format **YYYY-MM-DD**.

~E, ^E

The full path name of the executables directory of SPFLite; the directory into which SPFLite was installed. A sample value for ~E would be "C:\Program Files\SPFLite".

~F, ^F

The simple file name of the original file without the path or extension. When submitting from the clipboard, this will be an empty string. For file name **C:\ONETWO.TXT**, **~F** contains **TWO**.

~I, ^I

The full path and file name of the temporary file containing the selected contents of the current **EDIT** (or **BROWSE** or Clipboard) session from which the **SUBMIT** primary command was issued. The **~I** code must be present on the **SUBMIT** prototype command line for the submit process to operate according to its intended purpose. It would be considered unusual for a **SUBMIT** prototype command not to contain the **~I** code, but SPFLite does not enforce such usage. Names for temporary submit files have the general form **SUBMIT_JOB12345.TXT**.

~J, ^J

A job ID, formatted as **JOBnnnnn**. **JOB** is constant and **nnnnn** will be an incrementing number starting at 1 and incremented by 1 for every submitted job. The **~J** counter is kept in the **SPFLite.INI** file as the persistent value **JobNumber**, and is installation-initialized to 1. This provides a means of assigning every job a unique job number and a unique file name for the temporary files that get created.

~K(ISODATE), ^K(ISODATE)

The current date when batch process submitted, in the format **YYYY-MM-DD**.

~K(ISOTIME), ^K(ISOTIME)

The current time when batch process submitted, in the format **HH:MM:SS**.

~K(primitive), ^K(primitive)

Any of the keyboard primitives which emit data may be used, as well as any key definitions which contain either un-bracketed strings (like primary commands) or those bracketed in [] brackets (i.e. simple keyboard text strings. Keys are specified by their official name, which can be seen when the key is selected in the **KEYMAP** dialog. For example, **~K(F2)** substitutes the string assigned to the F2 key. For key variations, prefix the keyname with the letters S,C or A (Shift, Control or Alt) in any combination followed by a – dash: **~K(S-F2)** for Shift-F2; **~K(CA-Home)** for Control-Alt-Home.

~M, ^M

Provides extended file Mode information. This submit code may be passed to any desired program, but is intended for use with a future version of the SPFSUBMIT.EXE program. The Mode information can be used to handle cases where **SUBMIT** is used with files of non-standard file formats, character sets, etc. This information is taken from the current Profile values. When a new file is being submitted, or a file is submitted from the File Manager using the **J** line command that has no existing profile, the **DEFAULT** profile's values are used. When the **~M** or **^M** code is used, a set of 7 command-line parameters are generated, in the following format:

-SUB:m -EOL:eee -LEN:nnn -FMT:fff -SRC:sss -TAB:ttt -EBC:D

where

-SUB:m is a Submit method code.

J = submitted from File Manager with **J** line command

S = submitted from edit session with **SUBMIT** command

-EOL:eee is the current EOL setting, such as CRLF

-LEN:nnn is the current LRECL setting, which is 0 for a standard text file

-FMT:fff is the current RECFM setting, which is U for a standard text file

-SRC:sss is the current SOURCE setting, which is ANSI for a standard text file

-TAB:ttt is the current XTABS setting

~N, ^N

The full, complete name of the original file, path included. For file name **C:\ONE\TWO.TXT**, **~N** contains **C:\ONE\TWO.TXT**.

Note: Since SPFLite has opened, and has access to the original file during the **EDIT** or **BROWSE** session, it could present run-time issues if any user program or script tried to open and modify the same original file while SPFLite still had 'ownership' of it. It is possible such usage might succeed, but SPFLite cannot guarantee the outcome under such circumstances. Users are advised to treat the **~N** code as an "information-only" value and not attempt to open the file during the time SPFLite has the underlying **EDIT** file open.

~P, ^P

The path of the directory containing the original file. When submitting from the clipboard, this will be an empty string. For file name **C:\ONE\TWO.TXT**, **~P** contains **C:\ONE**.

~R, ^R

When specified, **~R** is expanded to the full path name of a temporary file (a "Response file") used to provide job results status from the user's job submission procedure. This temporary file name will have an SPFLite-provided name incorporating the current job number, and an extension of **.TXT**. These temporary files are located in the user-specified **SUBMIT** working directory. When the **SUBMIT** prototype command contains the **~R** code, SPFLite will monitor the temporary Results file for any changes to it, and will display the lines of this file in a popup display. The user is responsible for formatting and writing a brief, meaningful message to this file. Use of the **~R** response file code is optional; when omitted; SPFLite makes no attempt to monitor for a Results file. One source for creating the Results file is the standard output of the **SPFSUBMIT** utility program if redirected to a text file. Note that SPFLite will create this as an empty file when **~R** or **^R** are specified; and then it monitors that file name for any file-create or file-update activity. Names created as response-file names have the general form **SUBRESULTS_JOB12345.TXT**.

~S (name) , ^S (name)

When specified, the string value of the defined SET variable **name** is substituted. See the [SET](#) primary command for more information on defining SET variables.

~T, ^T

The current directory time of the original file, in the ISO format **HH:MM:SS**.

~X, ^X

The file extension of the original file, including the leading dot, or else an empty string if file has no extension. When submitting from the clipboard, this will be an empty string. For file name **C:\ONE\TWO.TXT**, **~X** contains **.TXT**.

~Z, ^Z

The contents of the **SUBARG** string. A **SUBARG** string is defined on the PROFILE-level for a given file type, and is defined by edit primary command **SUBARG**. **SUBARG** takes one operand, a *submit argument* string. This value must be quoted if it contains embedded blanks; any of the 3 quote types are permitted. The contents of **SUBARG** may be displayed by issuing [SUBARG](#) with no operand, or by issuing the **PROFILE** command to see this value along with all other profile settings. When a **SUBARG** value for a given file type is undefined, the value of **~Z** is **OFF**, as a 3-character upper case value. The value is undefined by default and can be undefined by issuing **SUBARG OFF**. Thus, **OFF** (in any upper or lower-case spelling) is reserved. Any other **SUBARG** value may be set, and is accepted as-is (the value is not upper-cased). Note that if you actually wish to use **OFF** and **ON** as values, that is possible, except that **OFF** will always appear in upper case, while an **ON** value, if used, appears just as you enter it on the **SUBARG** command. It is not possible to define the **~Z** code as an empty string. The value **OFF** is handled the way it is to make checking for it easier in batch scripts.

Note: What would the **SUBARG ~z** value be used for? The submit edit file's extension is available from the **~x** code, so that can be used to tailor a submit script based on file type. Suppose a *group* of related files had some common requirement; then, the file type alone might not be enough to make the required distinction. Or, perhaps files are being submitted that do not have a file extension. The **SUBARG** value provides a means to tailor the submit process in these special circumstances.

See [SUBARG - Set Default SUBMIT Argument](#) and [Working with the SUBMIT Command](#) for more information.

Examples

To submit all of the data as a batch process:

SUBMIT

To submit lines between labels .START and .END as a batch process:

SUBMIT .START .END

To submit all of the data to a batch process, with the arguments "ONE" and "TWO THREE":

SUBMIT [ONE "TWO THREE"]

To submit only non-excluded lines as a batch process:

SUBMIT NX

To submit only excluded lines not having tag :T between lines .A and .B as a batch process:

SUBMIT X :\T .A .B

SWAP - Switch to a Selected File Tab

Syntax

SWAP	[PREV NEXT PRIOR FIRST LAST HOME LIST]
-------------	---

Operands

PREV	Switch to the tab to the left of the current tab
NEXT	Switch to the tab to the right of the current tab
PRIOR	Switch to the tab that was active prior to the current one
FIRST	Switch to the left-most file tab
LAST	Switch to the right-most file tab
HOME	Switch to the File Manager tab. The file list that is displayed is the one last in effect.
LIST	Switch to Open Files File List within the File Manager. See discussion below.

Description

The active edit tab can be changed with the mouse by left-clicking on the desired file tab at the top of the SPFLite window.

SWAP commands can be assigned to mapped keys, to allow keyboard-controlled switching of the active tab.

To conform to the way in which most Windows-based applications do this, you would map the **SWAP NEXT** command to Ctrl-TAB, and map the **SWAP PREV** command to Shift-Ctrl-TAB.

The options **FIRST**, **LAST**, **HOME** and **LIST** are available.

The SWAP PRIOR command allows you to toggle between two files, when you must frequently switch between one and the other. The easiest way to do this is as follows:

- Select a key you wish to use for the **SWAP PRIOR** command, and map that command to your key in KEYMAP
- Left-click on your first file of interest, then on your second file of interest
- From that point on, the key you have assigned to **SWAP PRIOR** will toggle between those two file tabs.

The **SWAP** command should not be confused with swapping lines or swapping sections of text. For these actions, see the [W / WW - Swap Lines](#) command and the [\(Swap\)](#) function in [List of Keyboard Primitives](#).

The SWAP LIST command

In ISPF, the **SWAP LIST** command brings up the "Active ISPF Logical Sessions" dialog. The closest equivalent to this in SPFLite is the **Open Files** File List display in File Manager, and that is what will be displayed when

SWAP LIST is issued. The same list will be displayed if you issue a [RECALL OPEN](#) command. If the File Manager list is not being display when **SWAP LIST** is used, the File Manager is displayed as with **SWAP HOME**, and then the **Open Files** File List brought up.

Note about ISPF SWAP

The IBM ISPF **SWAP** command and the SPFLite **SWAP** command perform similar, though not identical functions. **SWAP PREV** and **SWAP NEXT** operate essentially the same.

TABS - Turn Tabs On or Off

Syntax

TABS	[<u>ON</u> OFF]
-------------	---------------------

Operands

ON | OFF The desired TABS status

Abbreviations and Aliases

TABS can also be spelled as **TAB**

Description

The [TABS line command](#) allows you to display or alter the column positions to be used for tabbing

However, having to clear and/or re-establish tab positions every time you wanted to work with (or without) them would be burdensome.

The [TABS primary command](#) enables or disables tab support, without altering the currently-saved column settings previously established by the TABS **line** command.

The **ON/OFF** value of the [TABS primary command](#) is stored as part of the PROFILE options which are maintained individually by file type.

TAG - Alter Tag Status of a Range of Lines

Syntax

```

TAG          [ :tagname ]
              { search-string [ NF ] }
              [ start-column [ end-column ] ]
              [ ON | OFF | TOGGLE | ASSERT | SET ]
              [ FIRST | LAST | NEXT | PREV | ALL ]
              [ PREFIX | SUFFIX | WORD | CHAR ]
              [ line-control-range ]
              [ color-selection-criteria ]
              [ X | NX ]
              [ U | NU ]
              [ MX | DX ]
              [ TOP ]

```

Operands

:tagname

The tagname to be manipulated by this command.

Note: Although operands can be entered in any order, **:tagname** may appear twice in the command, as the tagname to be manipulated, and as one of the standard sub-operands of a line-control-range. In this command the first or leftmost one detected will be assumed to be the **:tagname** being manipulated, the second or rightmost, when two are entered, will be considered part of the line-control-range.

search-string

The search string that identifies the lines to be processed. **Note:** A search-string on a **TAG** command is just like a search-string on a **FIND** command, which means the same kinds of SPFLite string types are permitted, such as **C**, **T**, **X**, **P** and **R** strings.

start-column

Left column of a range (when end-column present) within which the search-string value must be found. If no end-column operand is present, then the search-string operand must be found starting in start-col.

end-column

Right column of a range (when start-column present) within which the search-string value must be found.

NF

Requests Not-Found search mode. It changes the search from looking for lines which **contain** the string to one which searches for lines which **do not contain** the string.

ON | **OFF** |

Specifies the action

TOGGLE |

ON

Will set the specified tag on the line.

ASSERT |

OFF

Will clear the specified tag on the line.

SET

If **:tagname** is specified, it will clear the tagname **only** if the existing tag on the line matches the one specified.

If **:tagname** is NOT specified, it will clear any existing tagname on

	the line.
TOGGLE	<p>If :tagname is NOT specified, it will clear any existing tagname on the line.</p> <p>If :tagname is specified, it will clear the tagname only if the existing tag on the line matches the one specified.</p> <p>If :tagname is specified, and no existing tagname is present, it will assign :tagname to the line.</p>
ASSERT	<p>If :tagname is specified, it will clear the tagname only if the existing tag on the line matches the one specified AND the search-string is not found in the line.</p> <p>If :tagname is NOT specified, AND the search-string is not found in the line it will clear any existing tagname on the line.</p>
SET	<p>SET requires a tagname and search-string. If search string is found on the line (or, not found, if the NF option is used), then the line is assigned the tagname.</p> <p>If search string is not found on the line (or, is found, if the NF option is used), then the line is cleared of any existing tags.</p> <p>SET may thus be used to assign a tag to a line range without have to pre-clear any existing tags in a separate step.</p>
FIRST	Starts at the top of the specified range and searches ahead to find the first occurrence of search-string.
LAST	Starts at the bottom of the specified range and searches backward to find the last occurrence of search-string.
NEXT	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search-string. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search-string.
ALL	Starts at the top of the data and searches ahead to find all occurrences of search-string.
PREFIX	Locates search-string at the beginning of a word.
SUFFIX	Locates search-string at the end of a word.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-alphanumeric characters
CHAR	.Locates search-string as-is, regardless of what precedes or follows it.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.

color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" . Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be examined, NX requests only non-excluded lines are to be examined. If neither X or NX are specified, all lines in the range will be examined.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
MX	MX requests that all lines which do contain search-string be excluded from the display following command processing. MX = Make excluded.
DX	DX requests that lines which do contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

TOGGLE can also be spelled as **TOG**

Description

The **TAG** command is used to affect tags based on a search string. **TAG** is basically modeled on the **FIND** command with some modifications.

The search arguments are used to locate specific lines, and the :tagname and **ON/OFF/TOGGLE** operands to specify how a tag is to be added or removed from the line.

Examples

TAG :BL ON " " 1 6 ALL

This will locate all lines containing blanks in columns 1 to 6 and set the tagname of **:BL** on those lines.

TAG OFF ALL .FROM .TO

This will clear all tags from the lines defined by the range of lines from **.FROM** to **.TO** inclusive.

TAG :A ON "A" 10 ALL

TAG :BOTH ON "B" 20 ALL :A

These two commands will first tag all lines with an "A" in column 10 with the tag :A. The second command will then examine all lines tagged with :A and those with a "B" in column 20 will be tagged with tag :BOTH.

See [Working with Line Tags](#) for more information.

TC - Title-Case a Range of Lines

Syntax

```
TC          [ line-control-range ]
           [ X | NX | ALL ]
           [ U | NU ]
           [ MX | DX ]
```

Operands

line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or NX are specified, all lines in the range will be processed.
U NU	Specifies a subset of the line range to be processed. U requests only User lines are to be processed, NU requests only non-User lines are to be processed. If neither U or NU are specified, all lines in the range will be processed.
MX	MX requests that all lines which are processed be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which are processed, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't alter Excluded status

Description

The lines processed will be converted to title-case. Title case means each word on the line will have its first letter set to upper-case and all other letters set to lower-case.

TOP - Scroll to Top of File

Syntax

```
TOP
```

Operands

None

Description

The **TOP** command will scroll the edit window to the top of the data file. **TOP** is an alias for **UP MAX**.

UC - Upper-Case a Range of Lines

Syntax

```
UC          [ line-control-range ]
           [ X | NX | ALL ]
           [ U | NU ]
           [ MX | DX ]
```

Operands

line-control-range The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)". Refer to that section of the documentation for details.

X | NX Specifies a subset of the line range to be processed. **X** requests only excluded lines are to be processed, **NX** requests only non-excluded lines are to be processed. If neither **X** or **NX** are specified, all lines in the range will be processed.

U | NU Specifies a subset of the line range to be processed. **U** requests only User lines are to be processed, **NU** requests only non-User lines are to be processed. If neither **U** or **NU** are specified, all lines in the range will be processed.

MX **MX** requests that all lines which are processed be excluded from the display following command processing.
MX = Make excluded.

DX **DX** requests that lines which are processed, which, if excluded, would normally be made visible, be left in their excluded status. **DX** = Don't change excluded status.

Description

The lines processed will have all text converted to upper-case.

ULINE - Mark User lines

Syntax

```
ULINE          [string]
                [ CHAR | WORD | PREFIX | SUFFIX ]
                [ start-column [ end-column ] ]
                [ line-control-range ]
                [ color-selection-criteria ]
                [ MX | DX ]
                [ X | NX ]
                [ ALL ]
                [ TOP ]
```

Operands

string	The string operand is optional. If provided, it requests a search, like FIND, to locate the lines to be marked as User lines. If not provided, then the line-control-range specification will be used to select lines.
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it.
WORD	Locates search-string when it is delimited on both sides by blanks or other non-Word characters.
PREFIX	Locates search-string at the beginning of a word.
SUFFIX	Locates search-string at the end of a word
start-column	Left column of a range (with end-column) within which the search-string value must be found. If no end-column operand, then the search-string operand must be found starting in start-col.
end-column	Right column of a range (with start-column) within which the search-string value must be found.
line-control-range	The range of lines which are to be processed by the command. Line control ranges provide a powerful tool to customize the range of lines to be processed. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ". Refer to that section of the documentation for details.
color-selection-criteria	A request for selection based on the highlight color of the search-string. Color requests provide another powerful tool to control search selection. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ". Refer to that section of the documentation for details.
ALL	All lines in the line range are processed.
X NX	Specifies a subset of the line range to be processed. X requests only excluded lines are to be processed, NX requests only non-excluded lines are to be processed. If neither X or

NX are specified, all lines in the range will be eligible to be processed.

MX	MX requests that all lines which DO contain search-string be excluded from the display following command processing. MX = Make Excluded.
DX	DX requests that lines which DO contain search-string, which, if excluded, would normally be made visible, be left in their excluded status. DX = Don't change Excluded status
TOP	Normally, at the completion of the command, the first, or only, line processed is highlighted (if it is on the current screen) or the screen is scrolled to the 2nd screen line (as ISPF does) if the line is not on the current screen. If TOP is coded, then the line is always positioned as the top line of the screen, regardless of its current location.

Abbreviations and Aliases

ULINE can also be spelled as **UU**

PREFIX can also be spelled as **PRE** or **PFX**

SUFFIX can also be spelled as **SUF** or **SFX**

WORDS can also be spelled as **WORD**

CHARS can also be spelled as **CHAR**

Description

ULINE will add the User line status to all lines which meet the specified criteria. This is an alternative method to using the **U** / **UU** line commands to mark lines as User Lines.

When an "ordinary" V line becomes a U line, a **|** vertical bar will appear in the "gap column".

Example uses of the **ULINE** command

To mark all lines from label **.FROM** to label **.TO** as User Lines:

```
ULINE ALL .FROM .TO
```

To mark all lines containing **(ABC)** as User Lines:

```
ULINE ALL "(ABC)"
```

To mark all lines containing the word **FRED** between columns 10 and 20, and which are highlighted in **RED**, as User Lines:

```
ULINE "FRED" WORD 10 20 RED ALL
```

For more information on User lines see "[Working with User lines](#)"

UNDO - Undo Changes

Syntax

```
UNDO
```

Operands

None

Description

UNDO will back out all edit changes made to a file since the last key was pressed that is considered an "attention" key. An attention key is a key that is mapped to the [\(Enter\)](#) keyboard function or is mapped to a primary edit command. Each time an "attention" event occurs when a file change has been made, it defines an undo-able change to the file.

Additional **UNDO** commands may be issued, up to the maximum level of **UNDO** commands that have been allowed for by the **SETUNDO** command; a maximum of 25 **UNDO** levels can be set.

See the [SETUNDO](#) command for more information.

Note: The effect of an **UNDO** can be reversed by the [REDO](#) command.

UNNUMBER - Remove sequence numbers

Syntax

```
UNNUMBER
```

Operands

UNNUMBER has no operands

Abbreviations and Aliases

UNNUMBER can also be spelled as **UNNUMB**, **UNNUM** or **UNN**

Description

The **UNNUMBER** command is used to remove existing sequence numbers from your file. Sequence numbers are removed by replacing them with an equal number of spaces. For records shorter than the right-most column of the sequence number field, they are padded with blanks on the right to make their length the same as that right-most column. This is a situation that would usually apply to newly-added or changed records that had no sequence number yet, or had one that was invalid.

If your sequence numbers are at the right side of your data and you UNNUMBER your file, the file will contain trailing blanks where the sequence numbers used to be. If you wish those trailing blanks to be trimmed off right away, you must do this manually after the UNNUMBER command completes. An easy way to do so is to place the line command **TR/** in the line-command area of line 1.

If you want trailing blanks at the end of each line to be trimmed off, but you can wait until your file is written, you can use the Profile option [PRESERVE OFF](#). Consult this command for more information.

If you use this approach, be aware that the trimming action performed by PRESERVE OFF occurs when the file is saved or when END is processed, but during a SAVE operation when you continue editing the file, any existing trailing blanks are not removed from the then-current edit session. If you need these trailing blanks removed immediately, you should the **TR/** line command.

See [Working with Sequence Numbering](#) for more information.

VIEW - View a File in Browse Mode

Syntax

```
VIEW           [ file-name | * ]
                  [ .Profile-name ]
```

Operands

file-name | * The name of the file to be viewed.

If a single * is entered, it requests the filename to be fetched from the current clipboard contents.

.Profile-name This optional operand allows you to specify an overriding Profile to be used for the View session. It simply consists of the name of the desired Profile, preceded by a . (period)

Abbreviations and Aliases

VIEW can also be spelled as **V**

Description

The VIEW command loads a file into the work space for browsing. If no file-name operand is specified, a standard Windows File Open Dialog will be presented for you to select a file.

In VIEW mode, you cannot SAVE the file, nor will AUTOSAVE cause the file to be saved. If you have altered the data, and wish to save the altered copy, you can use the CREATE or SAVEAS commands to save the data under a different file name.

Sometimes in View mode you may find that you have actually made changes (since it is SO similar to Edit) and then when you END the session, your changes were just thrown away. You can protect against this with the Options -> General setting for [**Warn on modified View file?**](#) This will cause a prompt if you are closing a modified View session and give you a chance to save the data before exiting.

In addition, when you modify a View file, the word View in the left-hand Status Bar box will be displayed as **View** to remind you it has been modified.

If the current working Tab contains data, a new Tab will be opened to hold the View session; otherwise the current Tab will be used.

Default Directory

When no operand at all or only a simple unqualified filename is provided for the VIEW command, the default directory used for searching for the file or for the file open dialog's starting directory will be determined as follows:

- If there is an active file being edited in the tab where the command is issued, then the Path for THAT active file is used as the default for the command.
- If there is NO active file [when the tab header displays (Empty)] then the current displayed directory of File Manager will be used.

Overriding Profile

When you wish to use a different Profile from the one indicated by the file's extension, simply provide the alternate profile name, preceded by a period, on the command line. For example to View MYSOURCE.BAS using the TXT profile the command would be

```
VIEW MYSOURCE.BAS .TXT
```

The use of the alternate profile is for the duration of this session **only**, it does not alter any permanent Profile processing.

VSAVE - Perform a Virtual Save of Data

Syntax

```
VSAVE
```

Operands

None

Description

You can perform a virtual save operation during the course of editing a file, using the primary command **VSAVE**. A **virtual save** will save the edit session to a temporary file, without committing the edit changes to the original permanent file you started out editing. If you are editing a critical file and experience a system outage, the VSAVE files can be used to resume editing, while if the edit ends normally, the VSAVE files are automatically deleted.

Note: if the requirement to maintain the original contents of the file as it was prior to the edit session is not important, then simply doing periodic [SAVE](#) commands would suffice. This can be automated using the [ACTION](#) function, review it for further details.

The intended purpose of a virtual save is to save works in progress without losing the contents of the original file. One might be performing some critical work, and may wish to frequently save it as a precaution against data loss. However, if the file is frequently saved in this way with the **SAVE** command, the original contents would be lost, in favor of saving a modified version of the file. It is possible for you to save the original file as a backup in another directory, or to make a copy under another name. Such methods have the drawback that management of such backups must then be done manually, a process that is often haphazardly done and can leave behind many spurious backup files that (once no longer needed) must be cleaned up or else they will take up disk space.

The VSAVE technique avoids the creation of spurious backup files, by internally managing temporary VSAVE files. You can **VSAVE** a file as many times as you wish, and the original file will remain unchanged. Once you are satisfied with your changes, you can **SAVE** the file, or **END** the edit session (assuming **AUTOSAVE** is enabled) and SPFLite will remove the temporary VSAVE file.

SPFLite creates a VSAVE directory underneath the SPFLite main directory. A VSAVE file name has a structured format, as follows:

VSAVE . *timestamp* .*fullpath*

where

VSAVE

is a constant prefix of the file

timestamp

is the time and date when this VSAVE file was first saved; the format of this timestamp is **YYYYMMDDHHmmSShh**, for year, month, day, hour, minute, second and hundredth of seconds.

Once a given edit session has been VSAVE'd, the timestamp for the VSAVE file does not change.

fullpath

is a modified form of the full path and name of the file being VSAVE'd. Because a full path name of a real file contains backslashes and a colon after the drive letter, such a string is not suitable as part of a simple file name. To accommodate this, the full path name of the subject file is modified so that the **** colon-backslash after the drive letter is converted into a single dot, and all other backslashes are converted to a dot. This convention is sufficient to ensure uniqueness of the VSAVE file name, while

giving it a readable name that will be recognizable when viewing the VSAVE directory.

For example, if a file named **C:\MYPATH\myfile.txt** were VSAVE'd, the VSAVE file might be called **VSAVE.2012012822375729.C.MYPATH.MYFILE.TXT**.

Other than the structured name and the directory where it's placed, there is nothing special about the contents of a VSAVE file. The data in a VSAVE file is byte-for-byte identical to a file saved using the SAVE command.

In the event of a system interruption that leaves behind undeleted VSAVE files, SPFLite will inspect the contents of the VSAVE directory during the next SPFLite startup. If existing VSAVE files are found that are more than two days old, these will be deleted. This means that under normal conditions, you will not need to worry about maintaining these files or about spurious VSAVE backup files needlessly taking up disk space.

At the same time, you will have a means to recover your work from the last **VSAVE** performed, if this becomes necessary. The recovery process would involve a manual inspection of the VSAVE directory, deciding which (if any) of the VSAVE files left behind are important enough to recover, and then to move these files to other directories as needed, at which point you most likely would rename such files to a more conventional file name format.

You do not need to set up or enable the VSAVE feature; it is always available to you. Your only decision is whether, and how often, you wish to issue the VSAVE command during your edit sessions.

In most cases, users will find the standard save commands **SAVE**, **SAVEAS**, **SAVEALL**, **CREATE** and **REPLACE** to be sufficient.

Where **VSAVE** is mostly likely to come into play is as follows:

- you need to edit large files
- you need to make many (possibly time-consuming) changes
- the changes are of critical importance
- system reliability is an issue
- your file is stored externally, such as on a network or an FTP server
- you wish to be prepared for any eventuality
- you prefer not to manually manage the creation of backup files
- you wish to have a means to frequently save interim edit changes
- and, you wish to be able to back out of the entire edit task if something goes wrong

These are all requirements of a mission-critical production environment, not the casual, every-day editing of a typical text file. These are the kinds of circumstances where **VSAVE** may benefit you.

WDIR - Open Windows Explorer

Syntax

```
WDIR
```

Operands

none

Description

Similar to the **DIR** primary command, **WDIR** will open the "containing directory" of the current edit file, but instead of displaying an SPFLite File Manager screen, a Windows Explorer dialog showing the containing directory will appear. This gives you access to any Windows-specific file handling tasks that are frequently done using a right mouse click to access a file's "context menus".

Once you are within an Explorer window, you can issue any context function that is accessible via the right mouse button. The available functions may vary, depending on any shell extensions, tools or software you may have added, but standard functions available with Windows include:

- Open
- Edit (with some other editor)
- Cut, Copy, Paste, Rename, Delete
- Properties (properties you could set include Read Only, Compression and Encryption)

The **WDIR** primary command works both in edit sessions and in the File Manager. In File Manager, **WDIR** will open an Explorer window for the directory shown on the File Path line.

Note this command cannot be used while working in [Multi-Edit](#) sessions. The reason for that is because a multi-edit session might consist of files from more than one containing directory.

For similar reasons, you cannot issue a **WDIR** primary command against a FILELIST. However, you can issue a **WDIR** line command against any individual file listed in a FILELIST or directory list.

XSUBMIT - Submit an external file

Syntax

XSUBMIT	file-name
----------------	------------------

Operands

file-name	The complete filename of an external file which is to be . All the normal processing done by SUBMIT will be performed using this file, instead of using data from the current edit session.
------------------	---

Abbreviations and Aliases

XSUBMIT can also be spelled as **XSUB**

Description

The **XSUBMIT** command performs all the same functions as the **SUBMIT** command. The difference is that instead of submitting the job from the data in the current edit session, it submits the data contained in an external file.

It is expected that the primary reason for using **XSUBMIT** is to submit jobs from within a programmable macro. Such a macro may be launched directly, or via the [SUBCMD](#) primary command, which redirects **SUBMIT** commands to perform some alternative command name, which may be the name of a macro; that macro, in turn, might issue an **XSUBMIT** command via the **SPF_CMD()** macro function.

All other processing is identical. For a full description of the processing involved, see the [SUBMIT](#) command, as well as [Working with the SUBMIT Command](#).

XTABS - Control Incoming Tab Characters

Syntax

XTABS	n
--------------	----------

Operands

n The desired size of the tab spacing

Description

The **XTABS** primary command defines the number of spaces that are used to replace a Horizontal Tab character (**HT = X'09'**) when these are found in the edit file. These tabs will be expanded to spaces, based on the Xtabs value.

When **n** is set to **0** (zero), Horizontal Tabs are not expanded.

The value is stored as part of the PROFILE options which are maintained individually by file type.

Introduction to Keyboard Primitives

Contents

- [*Introduction*](#)
- [*Clipboard Functions now support Named Private Clipboards*](#)
- [*Marked Text Blocks*](#)
- [*Repetition factors*](#)
- [*Movement Repetition*](#)

Introduction

In a key mapping definition, any string enclosed in () parentheses is treated as a keyboard primitive function. Keyboard primitives are those functions *other than* line commands or primary commands. In early versions of SPFLite, the only primitives available were simple functions like Enter, Tab, Delete, Insert, and Arrow keys. All those functions still exist (though some are renamed), and new ones have been added. The available keyboard primitive functions are listed below. Only primitive names predefined by SPFLite may be specified; you cannot create or add your own.

To assist in remembering these function names, a drop-down list is provided in the lower right corner of the KEYMAP dialog, shown as **Valid keyboard Primitives**. If you open this list and select an item by clicking on it, the value will be placed in the clipboard. You can then paste that value into the text box for your desired key. This saves time and avoids typing errors.

Clipboard Functions support Named Private Clipboards

Keyboard functions that involve the clipboard can take an option to specify a Named Private Clipboard. The format of this option is a / slash followed by the name of the private clipboard. For example, suppose a private clipboard of **myclip** exists. To paste from the Windows clipboard, you would use the function ([Paste](#)), but to paste from the **myclip** private clipboard, you would use ([Paste / myclip](#)). This technique works for every keyboard function that uses a clipboard.

Marked Text Blocks

Many of the primitive functions require a block of text to be highlighted (marked), either with the mouse or keyboard mark functions. For example, functions like (Copy) and (UpperCase) require a block of text to have been previously highlighted (marked) before the data is copied or changed to upper case. Now, if functions like these are used when the cursor is in the data area, but no data is actively being highlighted, the single character at the cursor position is used **as if** that character were highlighted. Applying functions to single characters is now easier, faster and more reliable.

For example, a (Copy) function (normally mapped to Ctrl-C) is requested with no area marked, then the single character located at the cursor location will be copied to the clipboard.

Repetition factors

You may specify a **repetition factor** for primitive functions. The repetition factor is specified as a decimal number just after the opening left parenthesis of the function, and followed by a colon.

Note there is also a **Movement repetition** provision available for the cursor movement primitives which is

actually more efficient for those functions, see Movement repetition. It does not preclude using the method directly below, it will simply perform the operation faster.

For example, the function **(Right)** will move the cursor one character to the right. If you wanted to move the cursor four positions to the right, you previously had to specify this as **(Right)(Right)(Right)(Right)**.

With a repetition factor, this can be simplified to **(4:Right)**.

It is also possible to use repetition factors to repeat literal text. For example, to generate twenty asterisk characters, instead of using

```
[*****]
```

you can now achieve the same effect with

```
(20 : [*])
```

and it will run faster. This feature will make it easy to define long strings that would be impractical to enter literally.

For example, to generate 80 asterisks, you could specify **(80:[*])**, which would be difficult to enter in the KEYMAP definition if you had to individually type all 80 asterisks manually; you might count them wrong, or the entire string might not even fit in the field.

Other than text repetition, repetition factors can only be used within primitive functions in **()** parentheses, not on un-enclosed primary commands or on the other types of enclosed keyboard items like **{ }** or **< >**.

Note: A repetition factor should only be applied to functions where it makes sense to use. For example, cursor movement makes sense if repeated, while repeating a **(Home)** command would be redundant, and repeating other commands like **(Ansi)** or **(Edit)** would produce undesirable behavior.

Movement Repetition

The cursor movement primitives **(Left)**, **(Right)**, **(Up)** and **(Down)** will support an option direct repeat operand. It is specified in KEYMAP when specifying the key by following the primitive name with a **/** and a numeric operand. e.g. **(Right/5)**

For example if the Right Arrow key were mapped to **(Right)** and Ctrl-Right Arrow mapped to **(Right/5)**, you would have the ability to move right normally with the Arrow key, and to move quickly by using the Ctrl-Arrow key.

Due to the way primitives are handled internally, a key mapped to **(Right/5)** will perform much faster than **(5:Right)**.

The **(Column/n)** function will move the cursor directly to column **n** of the data area. If **n** is omitted, **(Column)** will move the cursor directly to column 1.

Index to Keyboard Primitives

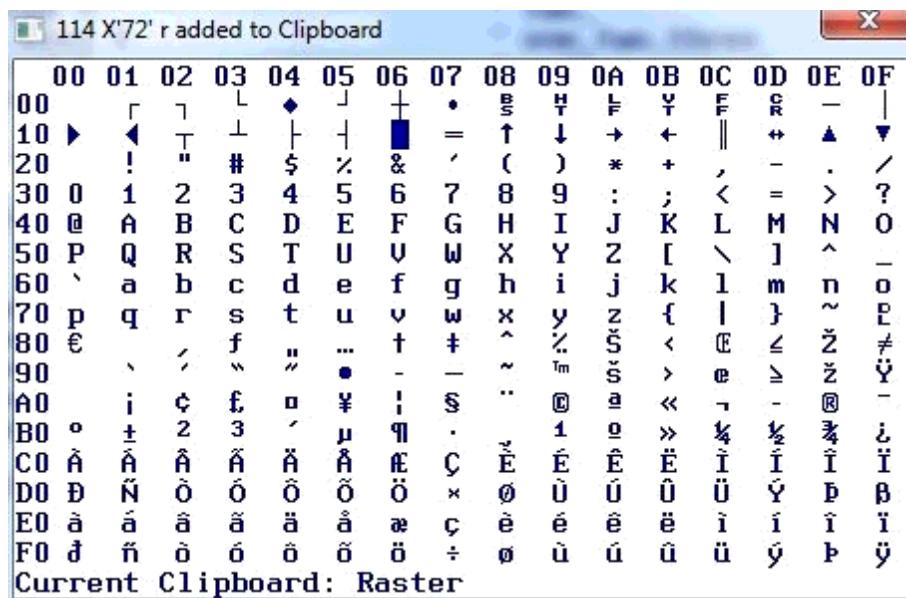
(BackSpace)	Move cursor left one position
(BackTab)	Move cursor left to previous tab stop
(BlockPaste)	Block-mode paste of clipboard contents
(Browse)	Use highlighted string as file name and open in Browse
(CharSet)	Display full 256 character set of edit font
(CharSetCol)	Same as above, but X / Y axes reversed
(ClipClear)	Clear the contents of the clipboard
(ClipDate)	Copy current date (Windows short format) into clipboard
(ClipISODate)	Copy current date (ISO format) into clipboard
(ClipISOTime)	Copy current time (ISO format) into clipboard
(ClipName)	Copy simple file name into clipboard
(ClipPath)	Copy fully-qualified file name into clipboard
(ClipTime)	Copy current time (Windows format) into clipboard
(Column)	Move cursor to a specific column number
(CondLineNo)	Cursor to next real data line.
(Copy)	Copy selected text into clipboard
(CopyAdd)	Copy (append) selected text into the clipboard
(CopyLCmd)	Copy line command data to the clipboard
(CopyPaste)	Copy or paste text to/from clipboard
(CopyPasteAdd)	Copy (append) or paste text to/from the keyboard.
(CopyPasteRaw)	Perform (CopyPaste) without copying EOL delimiters
(CopyRaw)	Perform (Copy) without copying EOL delimiters
(Cut)	Copy selected text into clipboard, then delete from edit file
(DataBackspace)	Perform (Backspace) while maintaining data alignment
(DataDelete)	Perform (Delete) while maintaining data alignment
(DataDeleteMark)	Perform (Delete) while maintaining data alignment only if an area is marked
(DataInsert)	Perform (Insert) while maintaining data alignment
(Date)	Paste current date (Windows short format) at cursor location
(Delete)	Delete selected text or character at cursor location
(DeleteMark)	Delete selected text. If no text selected, do nothing.
(Down)	Move cursor down one line
(Dup)	Duplicate a section of a line to one or more lines below it
(Edit)	Use highlighted string as file name and open in Edit
(EndOfLine)	Move cursor to right of right-most character on current line
(EndOfText)	Move cursor to right of right-most non-blank character on current line
(Enter)	SPFlite ENTER function, usually mapped to Enter or Right Ctrl key
(Enum)	Enumerate (sequence) text in decimal mode
(EnumHexLC)	Enumerate (sequence) text in hexadecimal mode using lower case a-f
(EnumHexUC)	Enumerate (sequence) text in hexadecimal mode using upper case A-F

(Erase)	Replace selected text with equal number of spaces
(EraseEOL)	Erase (delete) characters from cursor location to End of Line
(FindNext)	Find next selected text, or repeat prior find operation, going forwards
(FindPrev)	Find previous selected text, or repeat prior find operation, going backwards
(FirstLineCmd)	Move cursor to the First line command area on the screen
(FMCompact)	Toggles File Manager compact mode display
(Home)	Move cursor to left-most position of primary command line
(Insert)	Toggle Insert/Overtype mode and toggle INS/OVR on status line
(ISODate)	Paste current date (ISO format) at cursor location
(ISOTime)	Paste current time (ISO format) at cursor location
(JustifyC)	Center-justify selected text
(JustifyL)	Left-justify selected text
(JustifyR)	Right-justify selected text
(LastTab)	Move cursor to right-most defined tab position
(Left)	Move cursor left one column
(Lift)	Copy selected text to clipboard and then replace with equal number of spaces
(LineNo)	Move cursor to left-most position of sequence area
(LowerCase)	Convert selected text to lower case
(MarkDown)	Move cursor down and highlight/select text at cursor location
(MarkEnd)	Move cursor to last character on line and highlight/select text
(MarkLeft)	Move cursor left and highlight/select text at cursor location
(MarkRight)	Move cursor right and highlight/select text at cursor location
(MarkUp)	Move cursor up and highlight/select text at cursor location
(NewLine)	Move cursor down and to left-most position of sequence area
(NewLineNS)	Same as (NewLine) but will not cause scrolling at screen bottom
(Null)	No action taken
(PassThru)	Used so a key have its standard, Windows-defined behavior
(Paste)	Copy contents of clipboard to current cursor location
(PenBlue)	Change color of selected text to blue
(PenGreen)	Change color of selected text to green
(PenRed)	Change color of selected text to red
(PenStd)	Change color of selected text to standard text color
(PenYellow)	Change color of selected text to yellow
(PrtScrnClipboard)	Copy text image of screen to the clipboard
(PrtScrnLog)	Copy text image of screen to the SPFLite log file
(PrtScrnPrinter)	Copy text image of screen to the printer
(PrtTextClipboard)	Copy visible data lines to the printer
(Record)	Start/stop keyboard recording mode
(ResetInsert)	Turn Insert Mode off and return to Overtype Mode
(RestoreCursor)	Position cursor where last (SaveCursor) was issued
(RestoreInsert)	Restore Insert/Overtype mode from last (ResetInsert) or (SetInsert)
(Right)	Move cursor right one column
(SaveCursor)	Save current cursor position for later use by (RestoreCursor)

(ScrollDown)	Scroll the screen down while retaining relative cursor position on screen
(ScrollLeft)	Scroll the screen left while retaining relative cursor position on screen
(ScrollRight)	Scroll the screen right while retaining relative cursor position on screen
(ScrollUp)	Scroll the screen up while retaining relative cursor position on screen
(SentenceCase)	Convert selected text to sentence case
(SetInsert)	Set Insert Mode on
(Swap)	Used to swap text between two blocks
(Tab)	Move cursor right to next tab stop
(Time)	Paste current time (Windows format) at cursor location
(TitleCase)	Convert selected text to title case
(ToggleHome)	Toggle cursor between column 1 of data line and left-most non-blank position on line
(TxtHome)	Move cursor to column 1 of data line
(Txt.NewLine)	Move cursor down one line then to column 1 of data line
(Txt.NewLineNS)	Same as (Txt.NewLine) but it will not cause scrolling at screen bottom
(Up)	Move cursor up one line
(UpperCase)	Convert selected text to upper case
(View)	Use highlighted string as file name and open in View
(WordLeft)	Move cursor to prior word
(WordRight)	Move cursor to next word

List of Keyboard Primitives

- (BackSpace)** Move the cursor left one position, erasing the character directly left of the cursor. All characters to the right are shifted left one character.
- (BackTab)** Move the cursor left to the previous tab stop if within a text line and tabs are active, or to the beginning of the line if tabs are not active. If the cursor is at the beginning of the text line, the cursor moves to the beginning of the line number field. If the cursor is at the beginning of the line number field, the cursor moves to the last tab in the previous line. If on the top line number field, the cursor moves to the primary command line.
- (BlockPaste)** A block paste of the current contents of the clipboard will be performed with the current cursor location assumed to be the upper left corner of the block. The cursor must be in the text area, or the function is ignored.
- The (Paste) function will now perform both single-line and multi-line paste operations, and will serve the same purpose as (BlockPaste) when the clipboard contains a block. The (BlockPaste) function remains supported for compatibility purposes but is no longer required.
- (Browse)** This will take a highlighted string of text in the edit session (assuming it is a valid filename) and open the file in a new Browse tab, the same as is done by the **BROWSE** primary command. It is the user's responsibility to ensure that the highlighted text is a valid file name, or else a file open error is reported. When there is no active text highlighted, (Browse) has no effect.
- (CharSet)** (CharSet) opens a small window that displays the full 256 character set of your current working character set. (As specified by [SOURCE](#)).
- You may then use the mouse to select characters and they will be inserted into the clipboard. You can then return to the main edit window and Paste these characters as needed. This will enable you to enter special characters not present on your keyboard.
- Note:** When you view the Character Map using the (CharSet) function directly (but **not** when the Character Map is launched from the KEYMAP dialog), the Character Map window need not be closed once you place characters into the clipboard. This means, for example, that if you needed several different special characters to be inserted into several locations in your edit file, you can alternate between the edit screen and the Character Map window, by using Alt-Tab to switch between them.
- To ensure use of the correct characters, the CharSet window will be displayed using the same font you have chosen for editing in the Screen tab of SPFLite Global Options. See [Keyboard Customization and Keyboard Macros](#) for additional information on using the CharSet popup. The window will look like this:



(CharSetCol)	Same as (CharSet) except the X / Y axes are reversed. i.e. Columns are 00, 10, 20, 30, ... and Rows are 00, 01, 02, 03, ...
(ClipClear) (ClipClear / name)	Will clear the contents of the Windows clipboard or Named Private Clipboard.
(ClipDate) (ClipDate / name)	Will insert the current date, in normal Windows short format, into the Windows clipboard or Named Private Clipboard.
(ClipISODate) (ClipISODate / name)	Will insert the current date, in ISO format, into the Windows clipboard or Named Private Clipboard. ISO date format is YYYY-MM-DD.
(ClipISOTime) (ClipISOTime / name)	Will insert the current time, in ISO format, into the Windows clipboard or Named Private Clipboard. ISO time format is hh:mm:ss, 24 hr clock.
(ClipName) (ClipName / name)	Will insert the current filename and extension into the Windows clipboard or Named Private Clipboard.
(ClipPath) (ClipPath / name)	Will insert the current filename, including the drive/directory path, into the Windows clipboard or Named Private Clipboard.
(ClipTime) (ClipTime / name)	Will insert the current time into the Windows clipboard or Named Private Clipboard. Format is hh:mm:ss AM. 12 hour clock with trailing AM/PM.
(Column/nn)	Will move the cursor to column nn . The screen will scroll left/right as needed to bring the column into view.
(CondLineNo)	Move cursor to the Line Number field. If not on a data line, move to the next data line. If the file is Empty, place on the Top of Data line
(Copy) (Copy / name)	If there is currently highlighted, selected text on the screen, this will copy it to the clipboard.

name.

Example: **(Copy / myclip)**

(CopyAdd)

(CopyAdd / name)

If there is currently highlighted, selected text on the screen, this will **append** it to the clipboard.

(CopyAdd) acts like **(CopyRaw)**. That is, no end-of-line terminators are copied, even for multi-line (2D) highlighted text. Multiple lines are "run together" into a single text string.

To copy data to a Named Private Clipboard, enter a / slash followed by the clipboard name.

Example: **(CopyAdd / myclip)**

(CopyLCmd)

(CopyLCmd / name)

If the cursor is currently in the text area or line number area (for Edit screens) or on a file line (in File Manager) then this function will copy any current command in the line command area to the clipboard. If not on a valid line, or there is no line command present, then the clipboard is emptied.

Suppose you want the ability to 'propagate' a line command in a File Manager display. For example, suppose you want to edit several files. You could manually type in many **E** commands. But, it would be convenient to be able to type an **E** command in once, and then duplicate it as many times as needed. Here is an example key mapping macro to accomplish this. This key mapping example utilizes the **(CopyLCmd)** function with the **name** option. A suggested key to use with this is **Ctrl-Shift Enter**:

(Up)(CopyLCmd/Z)(Down)(Paste/Z)(LineNo)(Down)

This key mapping macro does the following:

1. Assume that you have entered a File Manager line command like **E** for Edit on some line, then move the cursor down to File Manager line command area on the next line, just under the **E**.
2. (Up) will temporarily move the cursor, back to where the **E** is.
3. (CopyLCmd/Z) will copy the line command **E** into the named clipboard **Z**. A named clipboard was used so as not to disturb the contents of the unnamed standard (Windows) clipboard, in case you had some important data in it.
4. (Down) moves the cursor back down just under where the original **E** command was.
5. (Paste/Z) pastes the line command (the **E** in this example) into the next line.
6. (LineNo) moves the cursor back to the beginning position of the line command area after the paste
7. (Down) moves the cursor under the second **E**, where you are ready to repeat the process.

When the **Ctrl-Shift Enter** key is set to automatically repeat, you would do this in practice to set up multiple line commands of **E**:

1. Put an **E** line command on a line
2. Cursor down to the next line
3. Pressing Ctrl and Shift, and press and hold the Enter key until you have as many copies of **E** and you need.

(CopyRaw)

(CopyRaw / name)

If there is currently highlighted, selected text on the screen, this will copy it to the clipboard.

(CopyRaw) acts like **(Copy)** except that no end-of-line terminators are copied, even for multi-line (2D) highlighted text. In raw mode, multiple lines are "run together" into a single text string.

To copy data to a Named Private Clipboard, enter a / slash followed by the clipboard name.

Example: **(CopyRaw / myclip)**

(CopyPaste)

(CopyPaste / name)

This is a dual-mode function. If there is currently highlighted, selected text on the screen, it acts in copy-mode to perform a (Copy) operation. If there is no current selected text, it acts in paste-mode to perform a (Paste) operation.

To copy or paste data to a Named Private Clipboard, enter a / slash followed by the clipboard name.

Example: **(CopyPaste / myclip)**

(CopyPasteAdd)

(CopyPasteAdd / name)

This is a dual-mode function. If there is currently highlighted, selected text on the screen, it acts in copy-mode to perform a (CopyAdd) operation. If there is no current selected text, it acts in paste-mode to perform a (Paste) operation.

(CopyPasteAdd) acts like **(CopyPaste)** except that, in copy-mode, no end-of-line terminators are copied, even for multi-line (2D) highlighted text. Multiple lines are "run together" into a single text string.

To copy or paste data to a Named Private Clipboard, enter a / slash followed by the clipboard name.

Example: **(CopyPasteAdd / myclip)**

(CopyPasteRaw)

(CopyPasteRaw / name)

This is a dual-mode function. If there is currently highlighted, selected text on the screen, it acts in copy-mode to perform a (Copy) operation. If there is no current selected text, it acts in paste-mode to perform a (Paste) operation.

(CopyPasteRaw) acts like **(CopyPaste)** except that, in copy-mode, no end-of-line terminators are copied, even for multi-line (2D) highlighted text. In raw mode, multiple lines are "run together" into a single text string.

To copy or paste data to a Named Private Clipboard, enter a / slash followed by the clipboard name.

Example: **(CopyPasteRaw / myclip)**

(Cut)

(Cut / name)

If there is currently highlighted, selected text on the screen, this will copy it to the Windows clipboard or Named Private Clipboard and will delete the selected characters, characters to the right on the line are shifted left.

(DataBackSpace)

This function will backspace in the same way as the standard **(BackSpace)** function, except that the data being "pulled left" by the backspace action is "delimited" by any span of two or more blank characters. Because of this, if you are backspacing in data that is formatted into columns, **(DataBackSpace)** will not disturb the column alignment, as long as two or more blanks separate the columns. A suggested mapping for this key is Ctrl-BackSpace.

See [Shifting Data](#) for more information.

(DataDelete)

This function will delete text in the same way that (Delete) does, except that the data being "pulled left" by the delete action is "delimited" by any span of two or more blank characters. Because of this, if you are deleting data that is formatted into columns,

(DataDelete) will not disturb the column alignment, as long as two or more blanks separate the columns. A suggested mapping for this key is Ctrl-Delete or Shift-Ctrl-Delete.

See [Shifting Data](#) for more information.

(DataDeleteMark) This function operates **(DataDelete)**, but **only** if a text area is marked. If there is no marked area, then the function does nothing.

A useful key mapping macro for the mouse, such as the Middle Mouse Button, is this:

**(DataDeleteMark)(DataInsert)(Paste)(RestoreInsert)(SaveCursor)(Enter)
(RestoreCursor)**

This will do the following:

1. Delete any highlighted text if present; otherwise no data is deleted. Deleted data does not 'pull' any data to the right of it.
2. Initiate Data Insert mode, so that the subsequent Paste does not 'push' data.
3. Paste any data from the clipboard into the line at the point where the cursor is.
4. Restore the Insert Mode to what it was before the Data Insert function was issued.
5. The remaining commands are necessary to complete the data insertion process while retaining the current cursor position

The actions above allow for the insertion of variable amounts of text, optionally replacing any highlighted data, in a manner very similar to how a word processor operates.

(DataInsert) This function will put the editor into Data Shift Insert Mode, comparable to the INS mode that occurs when the Insert key is pressed. Data Shift Insert Mode allows data to be inserted into lines in a way similar to the way that the Data Shift line command > operates. Because of this, if you are inserting data that is formatted into columns, **(DataInsert)** will not disturb the column alignment, as long as two or more blanks separate the columns. A suggested mapping for this key is Ctrl-Insert or Shift-Ctrl-Insert.

When Data Shift Insert Mode is in effect:

- non-blanks are shifted right as new data keys (including spacebar) are pressed, as usual
- when existing non-blanks are pushed to the right, and a span of two or more blanks follows the non-blanks, the span of blanks is shortened
- a span of blanks will never be completely removed by shortening it; there will always remain at least one blank
- the status indicator will show **INS** instead of INS or OVR
- pressing either of the the keys mapped to (Insert) or (DataInsert) will cause the status indicator to revert from **INS** back to OVR

See [Shifting Data](#) for more information.

(Date) Will Paste the current date, in normal Windows short format into the text at the current cursor location.

(Delete) If there is currently highlighted, selected text on the screen, the selected characters will be deleted. If no selected characters, then it will delete the character at the cursor location, characters to the right on the line are shifted left.

(DeleteMark) If there is currently highlighted, selected text on the screen, the selected characters will

be deleted. If no selected characters, then nothing is done.

(Down)	Will move the cursor down 1 line if the optional <i>nn</i> parameter is not provided, otherwise it moves the cursor down the specified number of lines.
(Dup)	<p>The (Dup) keyboard primitive operates on a highlighted line segment (one-dimensional area) or block (one-dimensional area) of text. The line segment above the first (or only) highlighted line segment is duplicated (copied) into the highlighted line or block. After the action is completed, the highlighting disappears. (Dup) also operates in Power Typing mode, and copies data from the line preceding the top/model line.</p> <p>It is an error to apply the (Dup) function to line 1 of a file, since there is nothing to copy/duplicate from. (Dup) does not use or alter the clipboard.</p> <p>Example: before:</p> <pre>000001 line *** one of file 000002 line two of file 000003 line /// three of file 000004 line four of file</pre> <p>An area is highlighted to indicate where the duplicated data is to be placed. Here, the *** string on line 1 is going to be copied to the next three lines after it, on lines 2, 3 and 4:</p> <pre>000001 line *** one of file 000002 line two of file 000003 line /// three of file 000004 line four of file</pre> <p>Now, press a key mapped to the keyboard function (Dup). (As a suggested mapping, assume that Ctrl-quote is used for this purpose. As usual, any desired key can be mapped to the (Dup) function.) When pressed, the *** string is copied down into the highlighted block:</p> <pre>000001 line *** one of file 000002 line *** two of file 000003 line *** three of file 000004 line *** four of file</pre>
(Edit)	This will take a highlighted string of text in the edit session (assuming it is a valid filename) and open the file in a new Edit tab. It is the user's responsibility to ensure that the highlighted text is a valid file name, or else a file open error is reported. If the highlighted text is a valid file name but the file does not exist, the message "File not found, created as empty file" is displayed, and the file will be created with the provided name, unless the CANCEL command is issued. When there is no active text highlighted, (Edit) has no effect.
(EndOfLine)	Will move the cursor to 1 character past the current end of line. If the line contains trailing blanks, it is placed after the last blank.
(EndOfText)	Will move the cursor to 1 character past the last non-blank character in the line.
(Enter)	The Enter key. You need a key mapped to (Enter) to use Primary and Line commands. If you fail to define at least one key as (Enter), SPFLite will require you to go back to the

KEYMAP and define one.

(Enum)	These primitives will trigger Enumerate processing.
(EnumHexLC)	For a full description of this feature, see " Working with Enumerations ".
(EnumHexUC)	
(Erase)	Will replace all highlighted characters with an equal number of spaces. It is an error to attempt to use (Erase) if no characters are currently highlighted. (Erase) will work both on a single line (1-D highlighting) and on a block (2-D highlighting), and will operate in Power Typing mode as well.
(EraseEOL)	Will erase all characters from the cursor location to the end of line (including the character at the cursor location).
(FindNext)	If there is currently highlighted text, (FindNext) finds the next occurrence of that text. If there is no currently highlighted text, (FindNext) finds the next occurrence of the string that was last searched for by (FindNext), (FindPrev), or by FIND, CHANGE or similar primary commands. If there is no text currently highlighted, and no prior search has been performed, an error is reported.
(FindPrev)	If there is currently highlighted text, (FindPrev) finds the previous occurrence of that text. If there is no currently highlighted text, (FindPrev) finds the previous occurrence of the string that was last searched for by (FindNext), (FindPrev), or by FIND, CHANGE or similar primary commands. If there is no text currently highlighted, and no prior search has been performed, an error is reported.
(FirstLineCmd)	Will move the cursor to column 1 of the first line command area on the screen
(FMCompact)	This primitive will act as a toggle for requesting a compact/non-compact version of the top part of the File Manager display. In compact mode, the display of Recent Files, Recent Paths, etc. is suppressed to provide more space for normal file names and less clutter.
(Home)	Will move the cursor to the leftmost position of the primary command line.
(Insert)	Will toggle the current Insert/Overtype status, which is displayed on the status line as INS or OVR.
(ISODate)	Will paste the current date, in ISO format, into the text at the current cursor location. ISO date format is YYYY-MM-DD.
(ISOTime)	Will paste the current time, in ISO format, into the text at the current cursor location. ISO time format is hh:mm:ss, 24 hr clock.
(JustifyC)	If there is currently highlighted text on the screen, it will be centered within the highlighted field. In Power Typing mode, text will be centered on every eligible line in the range of the Power Typing command. Where the total number of leading and trailing blanks is even, the blanks are evenly distributed to both sides of the text. Where the total number of leading and trailing blanks is odd, the blanks are distributed so that the right-hand side will have one more than the left-hand side.
(JustifyL)	If there is currently highlighted text on the screen, it will be left-justified within the highlighted field. In Power Typing mode, text will be left-justified on every eligible line in the range of the Power Typing command.

(JustifyR)	If there is currently highlighted text on the screen, it will be right-justified within the highlighted field. In Power Typing mode, text will be right-justified on every eligible line in the range of the Power Typing command.
(LastTab)	Will move the cursor to the last defined tab in the Tabs line, if Tabs are currently active.
(Left)	Will move the cursor left 1 character if the optional nn parameter is not provided, otherwise it moves the cursor left the specified number of characters.
(Lift)	Will copy all highlighted characters into the clipboard and then replace them with an equal number of spaces. Thus, the function is used to "lift" characters off the screen without the surrounding characters moving in any way. It is an error to attempt to use (Lift) if no characters are currently highlighted. (Lift) will work both on a single line (1-D highlighting) and on a block (2-D highlighting), and will operate in Power Typing mode as well.
(LineNo)	Will move the cursor to the leftmost position of the line number field if the cursor is currently on a data line.
(LowerCase)	If there is currently highlighted, selected text on the screen, it will be converted to lower case.
(MarkDown)	Will move the cursor down one line in text selection mode. If selection mode is not already set, it will turn on selection mode and highlight the current cursor location.
(MarkEnd)	Will move the cursor to the last text character in the line in text selection mode. If selection mode is not already set, it will turn on selection mode and highlight the text from the current cursor location to the last text character.
(MarkLeft)	Will move the cursor left one character in text selection mode. If selection mode is not already set, it will turn on selection mode and highlight the current cursor location.
(MarkRight)	Will move the cursor right one character in text selection mode. If selection mode is not already set, it will turn on selection mode and highlight the current cursor location.
(MarkUp)	Will move the cursor up one line in text selection mode. If selection mode is not already set, it will turn on selection mode and highlight the current cursor location.
(.NewLine)	Will move the cursor down one line and to the leftmost input column on the line.
(.NewLineNS)	Same as (NewLine) but will not cause screen scrolling when at screen bottom
(Null)	Will take no action when the key is pressed.
	When you have a key mapped to a line command, there is an implied (Enter) that is automatically inserted. In some cases, this implied (Enter) may not be what you wish. To suppress the implied (Enter), you can put (Null) after all other commands or functions.
(PassThru)	Will pass-through the normal Windows generated keyboard character for the key. When a key is mapped to (PassThru), it will perform the same action within SPFLite as it does outside of it.

(Paste)	Will paste the contents of the Windows clipboard or Named Private Clipboard at the current cursor location. If the clipboard contains multiple text lines, the Windows clipboard or Named Private Clipboard is pasted as a block, in the same way that (BlockPaste) does this.
(Paste / name)	The (Paste) function will perform both single-line and multi-line paste operations, and will serve the same purpose as (BlockPaste) when the clipboard contains a block. The (BlockPaste) function remains supported for compatibility purposes but is no longer required.
	To paste data from a Named Private Clipboard, enter a / slash followed by the clipboard name. Example: (Paste / myclip)
(Pen/colorname) (Pen/Std)	The area of text data that is currently highlighted (via mouse or arrow keys) is displayed using the specified <i>colorname</i> . <i>Colornames</i> are specified in the Options -> HiLites dialog.. Specify one of those names (or STD to restore text to the standard Foreground/Background colors). It is an error if there is no text data currently highlighted. See Working with Virtual Highlighting Pens for more information.
(PrtScrnClipboard) (PrtScrnClipboard / name)	Will send a text form image of the entire edit screen to the Windows clipboard or Named Private Clipboard.
(PrtScrnLog)	Will send (append) a text form image of the entire edit screen to the log file (SPFLiteScrPrt.LOG) in the SPFLite data directory.
(PrtScrnPrinter)	Will send a text form image of the entire edit screen to your default SPFLite printer.
(PrtTextClipboard) (PrtTextClipboard / name)	Will send only the actual data from the visible text lines to the Windows clipboard or Named Private Clipboard.
(Record)	Will start keyboard recording (if it is off) or stop recording (if it is on). If recording is was on and then is halted, and any keyboard activity was recorded, the resulting keyboard command string is copied to the clipboard. You may then switch to the KEYMAP dialog and paste the recorded string into whatever key combination you choose. See "Keyboard Customization and Keyboard Macros" for more details on creating keyboard macros.
(ResetInsert)	Will save the current status of the keyboard Insert Mode and then turn Insert Mode OFF regardless of its current setting.
(RestoreCursor)	Will restore the cursor to the same location saved by the (SaveCursor) primitive. If any of the following are true, the cursor is moved to the Command line. <ul style="list-style-type: none"> ○ No previous (SaveCursor) has been done ○ The previous (SaveCursor) was done when the cursor was not somewhere within a normal text line or its line number area. ○ Some action between (SaveCursor) and (RestoreCursor) has moved the location off the visible screen. That is, (RestoreCursor) will not cause screen scrolling to bring the location into view.

The (SaveCursor) and (RestoreCursor) functions can be used to perform action that normally would move the cursor away from its current location, in cases where you did not want it moved. For example, most primary commands will move the cursor in some

way. Here is an example of a keyboard macro that would insert an **A** line command on the current line, and then **PASTE** the contents of the clipboard after the current line:

{A} PASTE

However, this will end up moving the cursor away from the current line. To prevent the cursor from moving, you can 'wrap' the macro in (SaveCursor) and (RestoreCursor).

Note that if you use this approach, all of the macro 'elements' must be expressed in terms of the various values within the enclosures (), { }, [], and <>, because once a macro has an un-enclosed primary command, everything to the right of the command is considered to be command operands. To perform a primary command in this way, you have to explicitly "home the cursor", "type" the command as a literal, and "press Enter" using keyboard functions. Here is how the macro above can be rewritten. When the macro is performed, the PASTE operation will take place, and the cursor will be restored to the position it had before the PASTE was done.

(SaveCursor) {A} (Home) [PASTE] (Enter) (RestoreCursor)

Note 1: When line-commands are mapped to keys, there is an implied (Enter) that takes place when the line command is the last item in the key mapping. This implied (Enter) will be issued **only** if the line command appears last in the key definition. When you use (SaveCursor) and (RestoreCursor) functions, the main "command of interest" in the key mapping no longer appears last, and so the implied (Enter) will not take place. In such cases, you will have to specify an explicit (Enter) yourself.

For example, in another Help article, an example of toggling a line tag of **:M** was shown as a mapping of the line command **{ : :M}** to Ctrl-F2. Suppose you wanted to perform this function while also ensuring the cursor does not move in the process. You could try this:

(SaveCursor) { : :M} (RestoreCursor)

But because SPFLite does not resolve the line-tag toggling command until Enter is pressed, you will temporarily see **:M** in the line command area on the screen. This is harmless, but it looks unusual. The **:M** will be converted to a normal tag of **:M** as soon as you press Enter manually. This situation happens because (RestoreCursor) appears last in the key definition, and so the implied Enter never took place.

To ensure that the line-command is acted upon right away, while still restoring the cursor, you would need to rewrite the key definition like this:

(SaveCursor) { : :M} (Enter) (RestoreCursor)

Note 2: When adding an explicit (Enter) as shown above, be aware that SPFLite will do **everything** at the point you specify (Enter) as it would if you pressed the Enter key manually, including the processing of any pending primary edit commands. Be certain the actions that take place, and the conditions under which you use keys defined this way, are what you intended. Such key definitions may be considered "elaborate Enter keys", and should be treated as such.

(RestoreInsert)	Will restore the keyboard Insert Mode to the state it had prior to a previously issued (ResetInsert) or (SetInsert).
(Right)	Will move the cursor right 1 character if the optional nn parameter is not provided, otherwise it moves the cursor right the specified number of characters.

(SaveCursor)	Will save the current location of the cursor for use by the (RestoreCursor) primitive. See (RestoreCursor) for additional comments.
(SentenceCase)	If there is currently highlighted, selected text on the screen, it will be converted to lower case.
(SetInsert)	Will save the current status of the keyboard Insert Mode and then turn the Insert Mode ON regardless of its current setting.
(ScrollDown)	Scrolls the screen down while leaving the cursor at its current location. Note: When scrolling with the Mouse Wheel, holding down the Ctrl key causes 4 x Turbo Motion. To achieve the effect of "Turbo Mode" scrolling with the keyboard, a scroll function could be used more than once. For example, to make a scroll-down key scroll 4 lines at a time, you would map a key in the KEYMAP screen, using the repeat syntax, as (4:ScrollDown) . A suggested mapping for the scrolling functions would use Alt+arrow for regular scrolling, and Ctrl+Alt+arrow for turbo mode scrolling.
(ScrollLeft)	Scrolls the screen left while leaving the cursor at its current location. May be used in Power Typing mode. See Note above.
(ScrollRight)	Scrolls the screen right while leaving the cursor at its current location. May be used in Power Typing mode. See Note above.
(ScrollUp)	Scrolls the screen up while leaving the cursor at its current location. See Note above.
(Swap)	Invokes swapping or moving of text strings. For a full description of using this, see "Word Processing Support"
(Tab)	Will move the cursor to the next tab stop if in the text area and Tabs are active. Otherwise it will move to the next field on the same line. If no further fields, it will move to the first field on the next line.
(Time)	Will paste the current time into the text at the current cursor location. Format is hh:mm:ss AM. 12 hour clock with trailing AM/PM.
(TitleCase)	If there is currently highlighted, selected text on the screen, it will be converted to lower case.
(ToggleHome)	If the cursor is currently on column 1 of the data, it will move to the leftmost non-blank character of text data. If the cursor is currently on a column other than column 1 of the data, or in the sequence area, it will move to column 1 of the data. If the cursor is anywhere on a blank line, the cursor will move to column 1 of the data. If the cursor is located anywhere else, such as the primary command area or a non-data area, no action will occur.
(ToggleSelect)	This is the keyboard equivalent of left-clicking the Status Bar box containing the current

text selection values (or the word 'Select'). See [Working with Line Labels](#) for more details.

(TxtHome)	Will move the cursor to the leftmost visible column of the text data (column 1 of the data).
(Txt.NewLine)	Will move the cursor to the leftmost visible column of the text data of the next line (column 1 of the data). This is similar to the (NewLine) function except the cursor is placed in the text area rather than the line number area.
(Txt.NewLineNS)	Same as Txt.NewLine but will not cause screen scrolling at the bottom of the screen.
(Up)	Will move the cursor up 1 line if the optional <i>nn</i> parameter is not provided, otherwise it moves the cursor up the specified number of lines.
(UpperCase)	If there is currently highlighted, selected text on the screen, it will be converted to lower case.
(View)	This will take a highlighted string of text in the edit session (assuming it is a valid filename) and open the file in a new View tab, the same as is done by the VIEW primary command. It is the user's responsibility to ensure that the highlighted text is a valid file name, or else a file open error is reported. When there is no active text highlighted, (View) has no effect. Starting in version 8.1 of SPFLite, the (Browse) function and the BROWSE primary command operate in a strict non-modification mode. No changes of any kind can be performed on a browsed file, not even temporarily. To obtain the less-strict functionality performed by the (Browse) function and the BROWSE primary command that was present in prior versions, use the (View) function or the VIEW primary command. That is, you can think of (View) and VIEW as being the old Browse function that has simply been renamed. This change was made to enhance ISPF compatibility.
(WordLeft)	Will move the cursor left to the beginning of the current word (if within a word) or to the beginning of the previous word if already at the beginning of a word. Repeated use will continue by moving to the previous line.
(WordRight)	Will move the cursor left to the beginning of the next word. Repeated use will continue by moving to the next lines.

Supplied Fixed Fonts

Supplied Fixed Fonts

[Overview](#)

[AnonymousRegular](#)

[DejaVu Sans Mono](#)

[DejaVu Sans Mono Bold](#)

[PixelCarnage Monospace](#)

[ProggyClean](#)

[ProggySmall](#)

[ProggySquare](#)

[QuickType Mono](#)

[Raize](#)

[Raster and Raster15](#)

[Raster14, Raster13 and Raster12](#)

[RasterTTF](#)

[Triskweline](#)

[IBM3270](#)

[MS LineDraw](#)

Overview

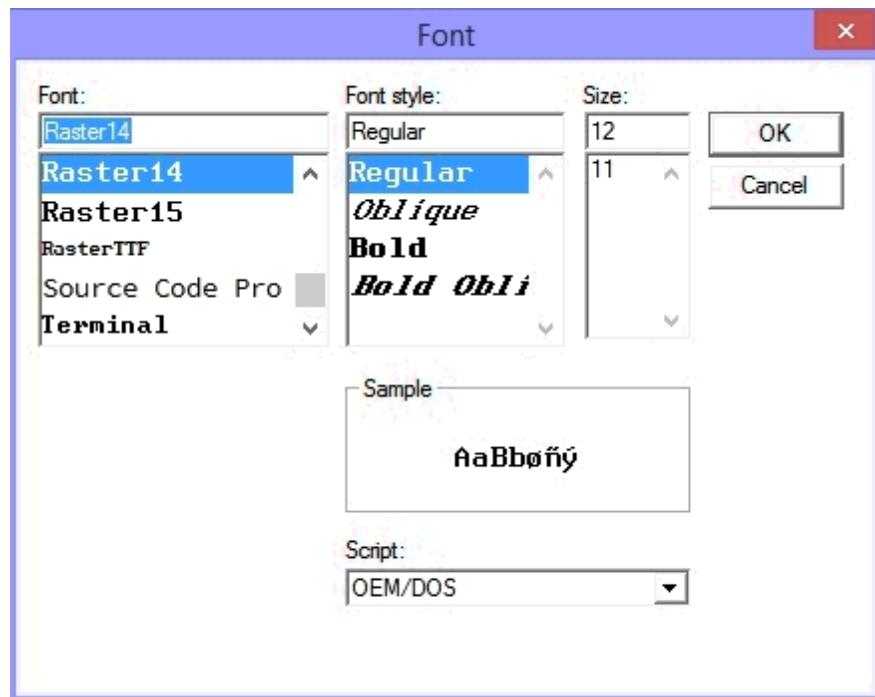
SPFLite can only use fixed pitch fonts for the edit screen. Although there are many free fonts available on the Internet, most of these are proportional fonts and are thus unusable. To supplement the standard Windows fonts like Courier and Terminal, the SPFLite website download page has an optional file containing a number of fixed fonts which you may freely use in your SPFLite configuration.

Shown below are screen shots of these fonts to help you choose. If you want to use one of them, download the [SPFLiteFixedFonts.ZIP](#) file and use the Windows Control Panel - Fonts application to install the ones you desire. Then, alter the Font settings in [Options - Screen](#) to specify the font.

The Raster/Raster15 or slightly smaller Raster14 fonts are particularly good in that they properly show many of the control characters (LF, CR etc.) and handle underlining properly if you use [HIDE](#) mode frequently. You also have fonts Raster13 and Raster12 available, when you still want a Raster font but screen space is at a premium.

Once you install the fonts, you can experiment with them within SPFLite by using [Options - Screen](#) to select a font. When you close the Options Dialog SPFLite will redisplay the screen in the selected font.

When you click on the Choose button, you will see a Windows Font selection dialog, like this:

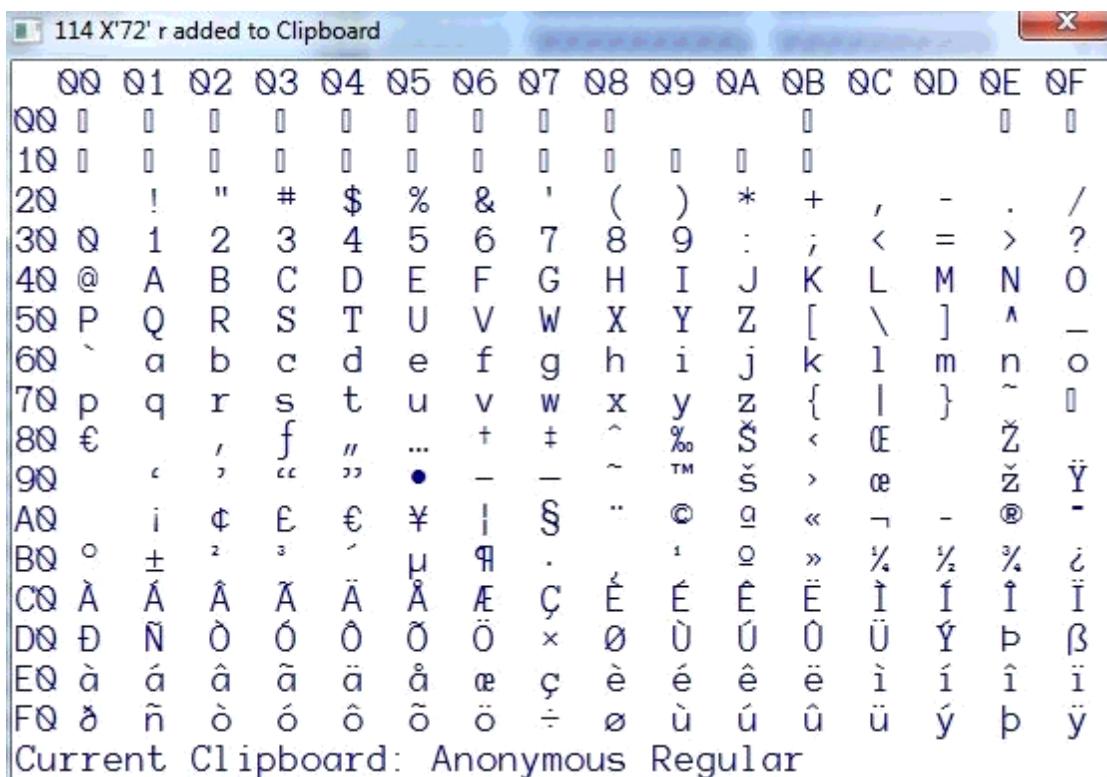


SPFLite causes only fixed-width fonts to appear in the font list.

At present, SPFLite only supports the **Regular** font style. If you pick one of the **Italic** or **Bold** styles, the selection will not work, and you will still get the Regular version of the font.

Note: If you are aware of other fixed-width fonts in the public domain that might be useful, let us know on the SPFLite user forums, including a link to where we can download and review the font. If it's a good one, we'll consider adding it to our list of supplied fonts, so everyone can benefit.

AnonymousRegular



DejaVu Sans Mono

ANSI - Left Click replaces Clipboard - Right Click ...																X
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00																00
10																10
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	0
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	50
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	70
80	€	,	,	f	"	"	..	†	‡	^	‰	Š	„	Œ	Ž	80
90	,	,	"	"	•	—	—	—	—	—	—	š	—	œ	ž	90
A0	i	¢	£	¤	¥	¦	§	“	”	©	¤	«	—	—	®	A0
B0	°	±	²	³	’	µ	¶	·	·	·	·	»	½	½	¾	½
C0	À	Á	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	C0
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Û	Ü	Ý	Þ	ß	D0
E0	à	á	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	E0
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	û	ü	ý	þ	ÿ	F0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Dec Hex Chr Len Current ANSI Clipboard:

111 6F 'o' 16 "DejaVu Sans Mono"

DejaVu Sans Mono Bold

ANSI - Left Click replaces Clipboard - Right Click ...

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
10	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	0
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	50
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	70
80	€	,	,	f	"	"	..	t	‡	^	‰	Š	š	Œ	Ž	80
90	,	,	,	,	,	,	,	,	,	,	,	™	š	>	æ	ž
A0	,	i	,	¢	£	¤	¥	,	,	,	,	©	¤	«	»	®
B0	,	±	,	²	,	,	,	,	,	,	,	,	,	,	,	
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Ï	C0
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	D0
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	ï	E0
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	F0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Dec Hex Chr Len Current ANSI Clipboard:
100 64 'd' 21 "DejaVu Sans Mono Bold"

PixelCarnage Monospace

101 X'65' e added to Clipboard

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	Ø	ø	♥	♦	♦	♦	♦	♦	♦	♦	Ø	♂	♀	♫	♪	*
10	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█	█
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	0
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	50
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	△
80	ç	é	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	ï	f
90	æ	Æ	â	Ã	ä	Å	Æ	ç	è	é	ê	ë	ì	í	ï	®
A0	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
B0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
C0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
D0	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
E0	ø	ß	π	Σ	σ	μ	γ	ξ	θ	Ω	δ	∞	ø	€	□	E0
F0	±	≥	≤	ƒ	J	÷	≈	°	.	.	∫	n	z	■	■	F0

Current Clipboard: PixelCarnage Monospace

ProggyClean

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	+	■	↳	↖	↙	↖	↗	↘	↖	↗	↘	↖	↗	+
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[]	^	~
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	ž
80	€	,	,	,	,	,	,	,	,	,	,	,	,	,	ÿ
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	i	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	À	Á	Â	Ã	È	É	Ê	Ë	Í	Ó	Ô
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	à	á	â	ã	è	é	ê	ë	í	ó	ô
F0	ð	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó

Current Clipboard: ProggyClean

ProggySmall

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[]	^	~
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	ž
80	€	,	,	,	,	,	,	,	,	,	,	,	,	,	,
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	i	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
D0	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
E0	à	á	â	ã	à	á	â	ã	è	é	ê	ë	í	ó	ô
F0	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ

Current Clipboard: ProggySmall

ProggySquare

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[]	^	~
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	ž
80	€	,	,	,	,	,	,	,	,	,	,	,	,	,	,
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	i	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	À	Á	Â	Ã	È	É	Ê	Ë	Í	Ó	Ô
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	à	á	â	ã	è	é	ê	ë	í	ó	ô
F0	ð	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó

Current Clipboard: ProggySquare

QuickType Mono

Due to an issue with the QuickType Mono font, you may not see the font listed when you click on the Choose button on the [Options - Screen](#) display. You can still use this font, by manually typing in the name **QuickType Mono** as the Font Name, and enter a Font Pitch as a decimal number. A font pitch of 10, 11 or 12 may be a good place to start for the font size.

111 X'6F' o added to Clipboard

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	o
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	ž
80	□	,	;	f	"	..	†	‡	~	™	§	š	š	ž	ÿ
90	.	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	i	,	¢	,	£	,	¤	,	¥	,	§	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	É	Ê	Ê	È	È	Í	Í
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ù	Û	Û	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	é	ê	ê	è	è	í	í
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ù	û	û	ý	þ
Current Clipboard: QuickType Mono															

Raize

101 X'65' e added to Clipboard

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	€	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	¤	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	o
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	í
80	€	,	,	,	,	,	,	,	,	,	,	,	,	,	,
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	i	,	¢	,	£	,	¤	,	¥	,	§	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	É	Ê	Ê	È	È	Í	Í
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ù	Û	Û	Ý	Þ
E0	à	á	â	ã	ä	å	æ	ç	é	ê	ê	è	è	í	í
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ù	û	û	ý	þ
Current Clipboard: Raize															

Raster and Raster15

These two fonts are identical, differing only in font name. Both are 15 pixels high by 9 pixels wide.

053 X'35' 5 added to Clipboard X

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	00
10	o	+	c)	4	5	6	7	8	9	h	l	Y	g	1	10
20	!	"	#	\$	z	&	'	()	*	;	;	;	;	;	20
30	0	1	2	3	4	5	6	7	8	9	I	J	Z	M	?	30
40	@	A	B	C	D	E	F	G	H	I	Y	J	L	N	0	40
50	P	Q	R	S	T	U	V	W	X	Y	i	z	j	l	^	50
60	'	a	b	c	d	e	f	g	h	i	z	k	m	n	-	60
70	p	q	r	s	t	u	v	w	x	y	z	l	l	l	l	70
80	€	†	‡	ƒ	„	„	„	„	„	„	„	„	„	„	„	80
90	■	,	,	,	,	,	,	,	,	,	,	,	,	,	,	90
A0	i	ç	£	¤	¥	₪	₪	₪	₪	₪	₪	₪	₪	₪	₪	A0
B0	o	±	2	3	4	5	6	7	8	9	o	o	o	o	o	B0
C0	À	Á	Â	Ã	Ä	Å	Æ	Œ	É	É	É	É	É	É	É	C0
D0	Ð	Ñ	Ò	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	D0
E0	à	á	â	ã	ä	å	æ	œ	é	é	é	é	é	é	é	E0
F0	đ	ñ	ò	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	F0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

Current ANSI Clipboard: Raster15

Raster14, Raster13 and Raster12

This is the same font as Raster/Raster15 but are shorter vertically (by eliminating white-space pixel rows) to allow a few more lines per screen.

052 X'34' 4 added to Clipboard X

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
10	▶	◀	↑	↓	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	▼
20	!	"	#	\$	¤	₪	₪	₪	₪	₪	₪	₪	₪	₪	₪	?
30	0	1	2	3	4	5	6	7	8	9	;	;	;	;	;	0
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	40
50	P	Q	R	S	T	U	V	W	X	Y	Z	l	m	n	-	50
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	60
70	p	q	r	s	t	u	v	w	x	y	z	l	l	l	l	70
80	€	‘	’	ƒ	„	„	„	„	„	„	„	„	„	„	„	80
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	90
A0	i	ç	£	¤	¥	₪	₪	₪	₪	₪	₪	₪	₪	₪	₪	A0
B0	o	±	2	3	4	5	6	7	8	9	o	o	o	o	o	B0
C0	À	Á	Â	Ã	Ä	Å	Æ	Œ	É	É	É	É	É	É	É	C0
D0	Ð	Ñ	Ò	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	D0
E0	à	á	â	ã	ä	å	æ	œ	é	é	é	é	é	é	é	E0
F0	đ	ñ	ò	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	F0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

051 X'33' 3 added to Clipboard															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	‘	’	“	”	◆	◆	•	↑	↓	↖	↖	↖	↖	↖	↖
10	▶	◀	!”	!”	!	!	=	‘	’	→	←	↖	↖	↖	↖
20															
30	0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	〔	〕	〔	〕
60	‘	’	‘	’	‘	’	‘	’	‘	’	’	‘	’	‘	’
70	p	q	r	s	t	u	v	w	x	y	z	〔	〕	〔	〕
80	€														
90															
A0	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦
B0	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦
C0	À	Á	Â	Ã	Å	Å	Å	Å	Å	Å	Å	Å	Å	Å	Å
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	å	å	å	å	å	å	å	å	å	å	å
F0	đ	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó
Current Clipboard: Raster 13															

050 X'32' 2 added to Clipboard															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	‘	’	“	”	◆	◆	•	↑	↓	↖	↖	↖	↖	↖	↖
10	▶	◀	!”	!”	!	!	=	‘	’	→	←	↖	↖	↖	↖
20															
30	0	1	2	3	4	5	6	7	8	9	0A	0B	0C	0D	0E
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
50	P	Q	R	S	T	U	V	W	X	Y	Z	〔	〕	〔	〕
60	‘	’	‘	’	‘	’	‘	’	‘	’	’	‘	’	‘	’
70	p	q	r	s	t	u	v	w	x	y	z	〔	〕	〔	〕
80	€														
90															
A0	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦
B0	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦	◦
C0	À	Á	Â	Ã	Å	Å	Å	Å	Å	Å	Å	Å	Å	Å	Å
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	å	å	å	å	å	å	å	å	å	å	å
F0	đ	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó
Current Clipboard: Raster 12															

RasterTTF

RasterTTF is the Raster font converted to TrueType format. RasterTTF has the same character set as Raster, but is intended primarily for printing files. RasterTTF has the advantage of being scalable, while Raster and Raster14 are crisper fonts for the screen. To print a mainframe-style "listing" file as 66 lines x 132 columns, you can use RasterTTF 11 Pitch, 0.5 top margin, 0.4 bottom margin, in landscape mode; there will still be room for a header and footer on the page. The sample below is RasterTTF in 16 pitch.

At present, this font is in Beta Mode as it does not appear to fully render on non-XP systems. The problem is only with a few special characters and a corrected font will be issued when the problem is resolved.

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	`	^	~	!	◆	—	+	*	≡	≡	≡	≡	≡	≡	≡	≡
10	▶	◀	↑	↓	—	—	—	=	↑	↓	→	←	—	▲	▼	?
20	!	"	#	\$	%	&	—	—	()	*	+	,	-	.	?
30	0	1	2	3	4	5	6	7	8	9	:	;	<	>	>	>
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	z	l	m	n	z
60	'	a	b	c	d	e	f	g	h	i	j	k	l	—	—	—
70	p	q	r	s	t	u	v	w	x	y	z	—	—	—	—	—
80	€	,	/	"	"	„	„	„	„	„	„	„	„	„	„	„
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	Ä	Å	Æ	É	Ó	É	Ó	É	Ó	É	Ó	É
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	ä	å	æ	é	ó	é	ó	é	ó	é	ó	é
F0	ð	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó

Triskweline

You are free to use this font, but don't ask us to pronounce it !

101 X'65' e added to Clipboard																
	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	`	^	~	!	◆	—	+	*	≡	≡	≡	≡	≡	≡	≡	≡
10	▶	◀	↑	↓	—	—	—	=	↑	↓	→	←	—	▲	▼	?
20	!	"	#	\$	%	&	—	—	()	*	+	,	-	.	?
30	0	1	2	3	4	5	6	7	8	9	:	;	<	>	>	>
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	z	l	m	n	z
60	'	a	b	c	d	e	f	g	h	i	j	k	l	—	—	—
70	p	q	r	s	t	u	v	w	x	y	z	—	—	—	—	—
80	€	,	/	"	"	„	„	„	„	„	„	„	„	„	„	„
90	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
A0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
B0	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,	,
C0	À	Á	Â	Ã	Ä	Å	Æ	É	Ó	É	Ó	É	Ó	É	Ó	É
D0	Ð	Ñ	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó	Ó
E0	à	á	â	ã	ä	å	æ	é	ó	é	ó	é	ó	é	ó	é
F0	ð	ñ	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó	ó

IBM3270

048 X'30' 0 added to Clipboard

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	▶	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	0
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\	^	0
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	o
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	4
80	€	B	f	■	■	■	■	■	■	■	■	■	■	■	■
90	À	à	Á	á	Ã	ã	Ä	ä	È	è	É	é	Í	í	Ï
A0	í	ó	ó	ú	ñ	ñ	ä	ö	ó	ó	ú	ú	ü	ü	ö
B0	±	²	³	‘	’	µ	·	·	·	·	·	·	·	·	·
C0	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À
D0	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ	Đ
E0	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à
F0	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë	ë

Current Clipboard: IBM 3270

MS LineDraw

119 X'77' w added to Clipboard

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
10	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
20	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	0
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\	^	0
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	o
70	p	q	r	s	t	u	v	w	x	y	z	{	}	~	□
80	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
90	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
A0	á	í	ó	ú	ñ	ñ	ä	ö	ó	ó	ú	ú	ü	ü	ö
B0	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
C0	└	└	└	└	└	└	└	└	└	└	└	└	└	└	└
D0	─	─	─	─	─	─	─	─	─	─	─	─	─	─	─
E0	α	β	Γ	π	Σ	σ	μ	τ	φ	•	Θ	Ω	·	∅	ε
F0	≡	±	≥	≤	∫	÷	≈	°	√	·	·	·	·	·	□

Current Clipboard: MS LineDraw

Automatic Colorization Files

Contents

- [*Introduction*](#)
- [*Auto Capitalization Support*](#)
- [*Setting up Automatic Colorization*](#)
- [*Automatic Colorization File Command Syntax*](#)
- [*Sample Colorize File*](#)

Introduction

SPFLite can display source files with keywords and delimiters automatically highlighted in various colors to assist in the readability of source code. To do this, SPFLite requires an Automatic Colorization file, which describes the particular source language (e.g. reserved words, delimiters used, comment indicators, etc.)

When the PROFILE option **HILITE AUTO ON** is chosen, (See the [HILITE](#) command) SPFLite will look for an Automatic Colorization file to use in the colorization process. If no matching Automatic Colorization file can be found, the file will be displayed as if colorization had not been requested. Automatic Colorization is based on the **file extension** of the file being edited.

For example, if editing a file MYSOURCE.BAS (a BASIC source program), SPFLite will look in the \SPFLite\AUTO\ data directory for an Automatic Colorization file called **BAS.AUTO** to use to control the colorize process. Note the file extension (**BAS**) is used as the filename appended with the file type of **.AUTO**. Similarly, for a C++ source program stored with an extension of .cpp, SPFLite would look for the file **CPP.AUTO**.

Many times, the source for one language is stored in a variety of file types (Header, Include, Macro, etc.) which would normally mean separate **AUTO** files for each file type, a real maintenance nuisance. The SPFLite Profile support allows you to 'point' a file type's profile to another already-existing profile. This way, all aspects of customization for a file type can be shared by other file types. This is performed by utilizing the [PROFILE](#) command option **USING** to point at the base, shared profile. Automatic Colorization support also follows the same sharing link and will use the **AUTO** Colorization file of the common profile.

The SPFLite web site has a down-loadable ZIP file containing a set of sample Colorization files for some of the common types. Note these files are **NOT** to be considered exact and complete for **any** of the languages provided. I created these very quickly from some keyword lists located on the web and they have received little or no real testing. The only one I'm fairly sure of is PowerBasic.AUTO since that is **my** language of choice.

As of Version 10, **AUTO** files no longer specify the actual colors to be used. The former **SCHEME** statements are no longer used, and are now simply ignored. Similarly the **NORMHI**, **NORMLO**, **LINEHI**, and **LINELO** are ignored.

The actual colors used by each **SCHEME** are now specified in [OPTIONS -> SCHEMES](#) - where it is now simpler to specify since it displays the result of your choices immediately.

This means the sample files are simpler to customize to your color preferences.

Auto Capitalization Support

As well as colorization support, SPFLite can be requested to capitalize all language keywords defined in the .AUTO file. For some computer languages, this can assist readability greatly. Note the uppercasing is done only during screen display, the actual text itself is not altered. Actual text changing can be done, See [AUTOCAPS](#) for more information on permanently altering the text itself.

To identify which keywords specified in the AUTO file are to be capitalized (it would be unusual to capitalize **every** keyword defined in the AUTO file), all that is needed is to change the WORD directive to AUTOCAPS..

For example, if the language keywords were previously controlled by a
WORD 1 FOR
it would now be specified as
AUTOCAPS 1 FOR

Setting up Automatic Colorization

- Locate the appropriate sample file for your language, copy it to the SPFLite \AUTO Data folder. If you are unsure of what directory this is, simply enter the [OPTION](#) command and at the bottom of the dialog box it will display the location of the SPFLite INI file. The **AUTO** colorize control files should be placed in the \AUTO folder within this directory. Ensure the base filename equals the normal extension used for the source. For example, if using PowerBasic as I do, you would copy and rename PowerBasic.AUTO to BAS.AUTO.
- Examine the SCHEME numbers used by the sample file and decide how this 'fits' with the SCHEMES you have specified in [OPTIONS -> SCHEMES](#). Then simply alter the Scheme numbers in the WORD statements to your choice in SCHEMES.

The structure is fairly simple (described below) and it should be easy to create or update these for any language. Should you update any of these, or create new ones for other languages, I encourage you to send them in to me; I will add them to the collection and make them available for all users.

Automatic Colorization File Command Syntax

Automatic Colorization files consist of either:

- comments (blank lines, or lines starting with a ; in position 1)
- definition statements

Definition Statements

Obsolete Statements

As of Version 10, the following Statement types are obsoleted. If present, they will be ignored:

SCHEME
NORMHI
NORMLO
LINEHI
LINELO

Note: A sample macro has been made available (**AUTOCnvt**) to simplify converting Pre-Version 10 AUTO files to the new format. Simply edit the AUTO file, and enter **AUTOCnvt** on the command line. The macro will not remove **any** comment lines, so if there are comment lines present describing your old SCHEME usage, you will need to manually review / delete them.

COMMENTn sn start
end

Up to Nine comment definitions are supported **COMMENT1** thru **COMMENT9**.

The **sn** operand refers to the scheme number used to display the comment. Each **COMMENTn** statement **may** reference different scheme's if desired, or the same scheme.

The **start** operand specifies the character (string) which indicates the beginning of comments.

The **end** operand can have one of three meanings:

1. **0** (zero) indicates the **start** string can begin anywhere in a line and the remainder of the line is a comment
2. **n** (positive number) indicates the **start** operand is to be recognized **only** in this position and the remainder of the line is a comment. (Like the COBOL comment indicator in position 7.).
3. **string** data indicates the comment starts with the **start** operand and ends with **this** operand. A typical example of this would be the comment markers /* or (* *).

Note: SPFLite does **not** support multi-line comments.

EXCLUDE col string

The **EXCLUDE** statement request that all text lines which match the specified criteria be **excluded** from further colorization processing. They will be displayed under control of the Normal Text Lo definition.

The *col* operand specifies a column number in which to look for the *string* operand; a match will cause the line to be excluded. If the *col* operand is coded as zero, the *string* operand will be scanned for anywhere within the text line.

You may code up to 10 **EXCLUDE** statements.

INCLUDE col string

The **INCLUDE** statement request that **only** text lines which match the specified criteria be passed on for further colorization processing. Lines which fail all **INCLUDE** statement selection will be displayed under control of the Normal Text Lo definition

The *col* operand specifies a column number in which to look for the *string* operand; a match will cause the line to be included. If the *col* operand is coded as zero, the *string* operand will be scanned for anywhere within the text line.

You may code up to 10 **INCLUDE** statements.

Note: If both EXCLUDE and INCLUDE statements are present, EXCLUDE processing is performed first, followed by INCLUDE processing.

QUOTED sn

The *sn* operand specifies the **Scheme** number to be used for the display of quoted literal strings.

NUMERIC sn

The *sn* operand specifies the **Scheme** number to be used for the display of literal strings which are not identified as a keyword, **and** contain only **numeric** characters. Numeric characters are considered to be 0-9, period and comma.

Note if the period or comma are also specified as delimiters in the **DELIMS** string, the DELIMS specification will take precedence. For example if the comma is a delimiter, the string 123 , 456 will be treated as a numeric string 123,

followed by a comma delimiter, followed by a numeric string **456**.

If the **NUMERIC** statement is not provided, Numeric strings will be displayed under control of the Normal Text Lo definition

ESCAPECHR x

Literals in some languages allow embedded special characters (including the single or double quote) within a literal if they are preceded by an 'escape' character, which basically says to ignore the next character as a delimiter. For example, if **ESCAPECHR** \ is specified, then the following string would be a valid quoted literal.

```
'What\'s the matter, didn\'t you understand'
```

DELIMS string

The **DELIMS** statement defines the delimiters which the language uses to separate operands and language elements. Normally the string would be something like **+-*/=()^<>[]{};** This string typically varies mostly due to what characters are allowable in variable names where sometimes characters like _ and . are allowed. Note that when the language uses relational operators like BASIC's <> or >= that the **individual** characters are what should be entered. Note that the space need not be specified, it is **always** a delimiter.

MIXEDCASE yes | no

Some computer languages are sensitive to the case of the keywords and can have duplicate keywords that have different meaning depending on their case. (e.g. void and Void in Java) If **MIXEDCASE YES** is specified, then the keyword search will be sensitive to the case of the words. If the default **MIXEDCASE NO** is specified, keywords will be searched for regardless of case.

WORD sn string **AUTOCAPS** sn string

The **WORD / AUTOCAPS** statements define the reserved words for the particular language.

The **sn** operand refers to the **Scheme** number (defined in [Options -> Schemes](#)) used to display the word. Different **Scheme** numbers may be used to classify reserved words into various categories.

When **AUTOCAPS** is used instead of **WORD**, it requests uppercasing of the specified keyword when it is detected.

The **string** specifies the actual reserved word. No embedded blanks are allowed, use multiple **WORDs** if needed.

Note: Each character entered in the **DELIMS** statement should be entered also with a **WORD** statement.

Sample Colorize File

```
; SPFLite colorize file for DOS Batch files
;
QUOTED 5
COMMENT1 1 REM 0           Either REM
COMMENT2 1 :: 1           or :: indicate comments
; Delimiters
DELIMS  () , . := @
WORD 6 (
WORD 6 )
WORD 3 ,
WORD 3 .
```

WORD	3	:	
WORD	3	=	
WORD	3	@	
WORD	4	aux	Reserved words
WORD	4	com1	
WORD	4	com2	
WORD	4	com3	
WORD	4	com4	
WORD	4	con	
WORD	4	lpt1	
WORD	4	lpt2	
WORD	4	lpt3	
WORD	4	prn	
AUTOCAPS	3	call	
AUTOCAPS	3	cd	
AUTOCAPS	3	defined	
AUTOCAPS	3	dir	
AUTOCAPS	3	do	
AUTOCAPS	3	echo	
AUTOCAPS	3	else	
AUTOCAPS	3	endlocal	
AUTOCAPS	3	equ	
AUTOCAPS	3	errorlevel	
AUTOCAPS	3	exist	
AUTOCAPS	3	exit	
AUTOCAPS	3	for	
AUTOCAPS	3	geq	
AUTOCAPS	3	goto	
AUTOCAPS	3	gtr	
AUTOCAPS	3	if	
AUTOCAPS	3	in	
AUTOCAPS	3	leq	
AUTOCAPS	3	lss	
AUTOCAPS	3	neq	
AUTOCAPS	3	not	
AUTOCAPS	3	nul	
AUTOCAPS	3	pause	
AUTOCAPS	3	set	
AUTOCAPS	3	setlocal	
AUTOCAPS	3	shift	

IBM ISPF Command Support

Following is a list of the normal IBM ISPF Primary and Line commands along with an indication of whether they are supported or not, and if so, the name of the equivalent SPFLite command.

Note: When a line command is longer than 1 character, the "block mode" name of the command is formed by repeating the last letter. For example, the command **UC** becomes **UCC**.

IBM Line Commands

IBM Line Command	Supported	SPFLite Command
(—Column Shift Left	Yes	<u>(</u>
)—Column Shift Right	Yes	<u>)</u>
<—Data Shift Left	Yes	<u><</u>
>—Data Shift Right	Yes	<u>></u>
A—Specify an "After" Destination	Yes	<u>A</u>
B—Specify a "Before" Destination	Yes	<u>B</u>
BOUNDS—Define Boundary Columns	Yes	<u>BNDS</u>
C—Copy Lines	Yes	<u>C/CC</u>
COLS—Identify Columns	Yes	<u>COLS</u>
D—Delete Lines	Yes	<u>D/DD</u>
F—Show the First Line	Yes	<u>F</u>
I—Insert Lines	Yes	<u>I</u>
L—Show the Last Line(s)	Yes	<u>L</u>
LC—Convert Characters to Lowercase	Yes	<u>LC/LCC</u>
M—Move Lines	Yes	<u>M/MM</u>
MASK—Define Masks	Yes	<u>MASK</u>
MD—Make Dataline	Yes	<u>MD</u>
O—Overlay Lines	Yes	<u>O/OO</u>
R—Repeat Lines	Yes	<u>R/RR</u>
S—Show Lines	Yes	<u>SI</u>
TABS—Control Tabs	Yes	<u>TABS</u>
TE—Text Entry	No	<u>none - see note 1</u>
TF—Text Flow	Yes	<u>TF/TFF</u>
TS—Text Split	Yes	<u>TS</u>
UC—Convert Characters to Uppercase	Yes	<u>UC/UCC</u>
X—Exclude Lines	Yes	<u>X/XX</u>

IBM Primary Commands

IBM Primary Command	Supported	SPFLite Command
AUTOLIST—Create a Source Listing Automatically	No	<u>none - see note 2</u>
AUTONUM—Number Lines Automatically	No	<u>none - see note 3</u>

AUTOSAVE—Save Data Automatically	Yes	<u>AUTOSAVE</u>
BOUNDS—Control the Edit Boundaries	Yes	<u>BOUNDS</u>
BROWSE—Browse from within an Edit Session	Yes	<u>BROWSE</u>
BUILTIN—Process a Built-In Command	No	<u>none - see note 4</u>
CANCEL—Cancel Edit Changes	Yes	<u>CANCEL</u>
CAPS—Control Automatic Character Conversion	Yes	<u>CAPS</u>
CHANGE—Change a Data String	Yes	<u>CHANGE</u>
COLS—Display Fixed Columns Line	Yes	<u>COLS</u>
COMPARE—Edit Compare	No	<u>none - see note 5</u>
COPY—Copy Data	Yes	<u>COPY</u>
CREATE—Create Data	Yes	<u>CREATE</u>
CUT—Cut and Save Lines	Yes	<u>CUT</u>
DEFINE—Define a Name	No	<u>none - see note 4</u>
DELETE—Delete Lines	Yes	<u>DELETE</u>
EDIT—Edit from within an Edit Session	Yes	<u>EDIT</u>
EDITSET—Display the Editor Settings Dialog	Yes	<u>OPTIONS</u>
END—End the Edit Session	Yes	<u>END</u>
EXCLUDE—Exclude Lines from the Display	Yes	<u>EXCLUDE</u>
FIND—Find a Data String	Yes	<u>FIND</u>
FLIP—Reverse Exclude Status of Lines	Yes	<u>FLIP</u>
HEX—Display Hexadecimal Characters	Yes	<u>HEX</u>
HIDE—Hide Excluded Lines Message	Yes	<u>HIDE</u>
HILITE—Enhanced Edit Coloring	Yes	<u>HILITE</u>
IMACRO—Specify an Initial Macro	No	<u>none</u>
LEVEL—Specify the Modification Level Number	No	<u>none - see note 6</u>
LOCATE—Locate a Line	Yes	<u>LOCATE</u>
MODEL—Copy a Model into the Current Data Set	No	<u>none - see note 7</u>
MOVE—Move Data	No	<u>none - see note 8</u>
NONUMBER—Turn Off Number Mode	No	<u>none - see note 3</u>
NOTES—Display Model Notes	No	<u>none - see note 10</u>
NULLS—Control Null Spaces	No	<u>none - see note 11</u>
NUMBER—Generate Sequence Numbers	No	<u>none - see note 3</u>
PACK—Compress Data	No	<u>none - see note 12</u>
PASTE—Move or Copy Lines from Clipboard	Yes	<u>PASTE</u>
PRESERVE—Enable Saving of Trailing Blanks	Yes	<u>PRESERVE</u>
PROFILE—Control and Display Your Profile	Yes	<u>PROFILE</u>
RCHANGE—Repeat a Change	Yes	<u>RCHANGE</u>
RECOVERY—Control Edit Recovery.	Yes	<u>SETUNDO</u>
RENUM—Renumber Data Set Lines	No	<u>none - see note 3</u>
REPLACE—Replace Data	Yes	<u>REPLACE</u>
RESET—Reset the Data Display	Yes	<u>RESET</u>

RFIND—Repeat Find	Yes	RFIND
RMACRO—Specify a Recovery Macro	No	none
SAVE—Save the Current Data	Yes	SAVE
SETUNDO—Set the UNDO Mode	Yes	SETUNDO
SORT—Sort Data	Yes	SORT
STATS—Generate Library Statistics	No	none - see note 9
SUBMIT—Submit Data for Batch Processing	Yes	SUBMIT
TABS—Define Tabs	Yes	TABS
UNDO—Reverse Last Edit Interaction	Yes	UNDO
UNNUMBER—Remove Sequence Numbers	No	none - see note 3
VERSION—Control the Version Number	No	none - see note 6
VIEW—View from within an Edit Session	No	VIEW

Note 1: The TE command was implemented by IBM to simplify large text entry on 3270 terminals. It is possible to enter text using the I line command, then post-processing it with other "text" line commands, to simulate the effect of TE.

Note 2: The PRINT and PRINT SETUP can be used to print a data file. AUTOLIST was not implemented because this command, implemented in the 1980's on IBM mainframes, supports a "hardcopy-centric" way of working, which modern PC users don't do.

Note 3: The Enumerate keyboard functions can be used to simulate some of the effect of AUTONUM, but would have to be applied manually as needed. AUTONUM in ISPF is intended to put sequence numbers in fixed columns of "card image" files, something that is rarely needed on PC files.

Note 4: Some of the functionality of BUILTIN and DEFINE is handled by the SET primary command.

Note 5: For COMPARE operations, an external Compare or DIFF utility can be used. A useful freeware DIFF program for Windows is called KDiff3.

Note 6: The LEVEL and VERSION commands were intended to store versions and modification level numbers in the "sequence field" of card-image data. For PC users, this functionality is better served by storing files in a source code control system of some type.

Note 7: The ISPF command MODEL is intended to copy 'prototype' definitions into a file, such as a standard calling sequence. It is possible to create a somewhat similar prototype file if the prototype file and your data file both have a file type for which STATE ON is in effect, and then use NOTE lines to define the prototype definition. As you fill-in your prototype to make it an actual part of your data file, you can use the MD (Make Data) line command to transform the NOTE lines into data lines.

Note 8: The ISPF command MOVE was not implemented because experience has shown that most mainframe users will first COPY an external file, and then delete it afterwards, after the file it was copied into is saved. Doing it that way prevents loss of information if there was an interruption between the MOVE and the subsequent SAVE. The MOVE command is thus a high-risk command. By not implementing MOVE, we have "erred on the side of caution" to help you avoid unrecoverable data loss.

Note 9: Presently, the only ISPF-compatible STATS information that SPFLite does not maintain is the number of data lines in the file. The remaining information that ISPF maintained is already present from information in the Windows directory entry, which appears on the File Manager display. The amount of detail shown in such displays is controlled by the File Manager tab in the SPFLite Global Options GUI.

Note 10: SPFLite does not have a Dialog Manager and does not support Models or model notes. It is possible

to create "model-like" files. See note 7.

Note 11: In ISPF it is necessary to manage the handling of 3270 NULL characters because it affects the transmission of data to and from the host, and can affect the ability to insert data into fields without being "locked out". In SPFlite, the equivalent of a 3270 "locked" condition will not happen, and so the NULLS command is not needed.

Note 12: The ISPF command PACK was intended to provide a form of data compression. This capability can be achieved directly in Windows through the use of data compression, applied as a property of an individual file or of its containing directory. Windows compression works better than ISPF PACK does, because unlike ISPF, Windows directly supports compressed files and does not require a decompress utility to access and use them.

Abbreviations

ANSI	ASCII			
BOTTOM	BOT			
BROWSE	B	BRO	BR	
BOUNDS	BOUND	BNDS	BND	BOU
CANCEL	CAN			
CHANGE	C	CHA	CHG	
CHARS	CHAR			
COLS	COL			
COMMAND	CMD	COM		
COMPRESS	CP			
CREATE	CRE			
CURRENT	CURR			
DELETE	DEL			
EDIT	ED	E		
EXCLUDE	EXCLUDED	EXC	EX	X
FAVORITE	FAVOURITE	FAV		
FIND	F			
HALF	H			
HILITE	HIGHLIGHT	HI		
KEYMAP	KEYS	KEY	KBD	
LABELS	LABEL	LAB		
LOCATE	L	LOC		
LOCK	LOCKED			
LRECL	LREC			
MACRO	MAC			
MAX	M			
NDELETE	NDEL			
NFIND	NF			
NEXCLUDE	NX			
NONUMBER	NONUMBR	NONUMB	NONUM	
NREVERT	NV			
NULINE	NU			
NUMBER	NUMB	NUM		
OPTIONS	OPTION	OPT		
PAGE	P			
PREFIX	PRE	PFX		
PROFILE	PROF	PRO		
PTYPE	PT			
REPLACE	REPL	REP		
RECALL	RC			

RENAME	no abbr.		
RENUM	REN		
RESET	RES		
REVERT	VV		
SAVE	SAV		
SETUNDO	SETU		
SPECIAL	SPE		
SUFFIX	SUF	SFX	
TABS	TAB		
TAGS	TAG		
TOGGLE	TOG		
ULINE	UU		
UNLOCK	UNLOCKED		
UNNUMBER	UNNUMB	UNNUM	UNN
USING	USE		
VIEW	V		
WORDS	WORD		

