

SPFLite3 - V3.1.24250
2024-09-06

Table of contents

Welcome to SPFLite	12
Support and Contacts	17
Major Features of SPFLite	18
Installing SPFLite	24
Creating a Portable Version	27
Differences between SPFLite and IBM ISPF	28
Use of the Mouse	32
Default Key Definitions	35
Customizing SPFLite	39
Options - General	41
Options - File Manager	50
Options - Submit	55
Options - Screen	58
Options - Keyboard	65
Options - Status	67
Options - Schemes	68
Options - Highlights	70
Options - Config	72
Keyboard Customization and Keyboard Macros	73
Key Mapping Overview	77
Using the KEYMAP Dialog	80
Keyboard Macros	91
Working with SPFLite	95
Starting and Ending SPFLite	96
Basic Edit Functions	102
AUTOFAV to add to File Lists	106
BACKUP & RESTORE	108
CFG File Maintenance	112
Clipboard, Cut and Paste	117
Command Keys and Substitution Strings	122
Command Pad	124
Crash Handling	125
Creating and Replacing Data Files	126
Delimited Literal	128
DIFF Compare	130
Edit Boundaries	138
Enumerations	140
Excluded Lines	144
Extended File Types	160
External File Changes	167
File Lists - FLISTS	169
File Manager	175
File Manager Primary Commands	184
File Manager Line Commands	186
File Profiles	193
FileOpen Startup Files	203
Finding and Changing Data	204

Find in Files	218
Including Another File	221
Instances and Configuration	222
Line Labels	226
Line Lengths	230
Line Tags	233
LINE Primary Command	243
MACROS	251
MultiEdit Sessions	251
NonWindows Text Files	260
NOTE and xNOTE Lines	267
Picture or Format String	273
Power Typing Mode	281
Print Screen Functions	295
Read Only Files	297
Regular Expression	301
Scrolling the Text	311
Shifting Data	313
SPFTest Program	319
SPLIT and JOIN Commands	321
STATE Saving	335
Status Bar Contents	338
SUBMIT Command	342
Tab Pages	348
Tabs and Column Markers	352
Tab Bounds Mode	355
Tracking Screen Positions	356
User lines	359
Virtual Highlighting Pens	368
Word and Delimiter Characters	373
Word Processing Support	375
Up Down Left Right Top Bottom	383
XFORM File transform macros	383
Line Commands	389
Rules for Entering Line Commands	392
Extended Line Command Modifiers	394
A - After Destination	397
AA - After Block	398
B - Before Destination	400
BB - Before Block	401
BNDS - Display Bounds Line	403
C / CC - Copy Lines	406
COLS - Display Columns Line	408
D / DD - Delete Lines	409
F - Display First Lines in Excluded Range	410
G / GG - Glue Lines Together	411
H / HH - HERE Destination Specification	413
I - Insert NewTemporary Blank Lines	414
J / JJ - Join Lines Together	415
L - Display Last Lines in Excluded Range	417

LC / LCC - LoweCase Lines	418
M / MM - Move Lines	419
MARK - Set Column Markers	422
MASK - Set the Insert line model	424
MD / MDD - Make Data Line	426
MN / MNN - Make Note Line	428
N - Insert New Permanent Blank Lines	430
NOTE - Insert NOTE or xNOTE Lines	431
O / OO - Overlay Lines	432
OR / ORR - Overlay Replace Lines	437
PL / PLL - Pad Lines to Length	439
R / RR - Repeat Lines	440
S - Show lines	441
SC / SCC - Sentence Case Lines	442
T / TT - Select Text Lines	443
TABS - Display TABS Line	446
TB / TBB - Text Break Lines	447
TC / TCC - Title Case Lines	449
TF / TFF - Text Flow a Paragraph	450
TG / TGG - Text Glue Lines	452
TJ / TJJ - Text Join Lines	454
TL / TLL - Truncate Lines to a Length	456
TM / TMM - Text Margin	457
TR / TRR - Trim Trailing Blanks	460
TS - Text Split a line	461
TU / TUU - Toggle User status of lines	463
TX / TXX - Toggle Excluded status of lines	464
U / UU - Mark User lines	465
UC / UCC - Upper Case Lines	468
V / VV - Revert User Line status	469
W / WW - Swap Lines	472
WORD - Display Valid Word Characters	474
X / XX - Exclude Lines	476
(Column Shift Left	477
) Column Shift Right	479
< Data Shift Left	481
> Data Shift Right	483
[Indent Shift Left	485
] Indent Shift Right	487
Primary Commands	489
Primary Command Notation Conventions	492
Line Control Range Specification	493
Color Selection Criteria Specification	496
Color Change Request Specification	499
ACTION - Request periodic SAVE	502
ADD - Add a text line	503
ALIGN - Align string on Column Boundary	504
ALL - Perform a Cmd. on all FM files	507
APPEND - Add text to end of line	507
AUTOBKUP - Control Creation of Backups	509

AUTOCAPS - Control AutoCaps Mode	510
AUTONAME - Specify Colorization file	511
AUTOSAVE - Control SAVE Action at END	512
BACKUP - Create a Backup of a file	513
BOM - Turn BOM writing ON / OFF	514
BOTTOM - Scroll to Bottom of File	515
BOUNDS - Set Edit Boundaries	516
BROWSE - Open a File for Read-Only	518
CANCEL - Cancel Edit Session	520
CAPS - Set Keyboard CAPS Mode	521
CASE - Set Default String Case Handling	522
CD - Change Directory	523
CHANGE - Change a Data String	523
CLIP - Open New File Tab with Clipboard Data	528
CLONE - Open an Unnamed Edit Using an Existing File	529
CLS - Clear / Close Debug Window	530
CMD - Execute Another Program or Command	531
COLLATE - Specify Display Character set	532
COLS - Control Visibility of Top Columns Line	533
COMPRESS - Compress Duplicate Strings	534
COPY - Include an External File	537
CREATE - Create an External File	539
CSV - Create Filelist CSV file	541
CRETRIEV - Conditional Retrieve	541
CUT - Cut Data to the Clipboard	542
DCB - Set File Characteristics	546
DELETE - Delete Selected Lines	550
DIFF - Compare two files	555
DIR - Display Folder in File Manager	557
DO - Invoke a KB Command file	558
EDIT - Open a File for Editing	559
EFT - Extended File Types	561
EMACRO - Set pre-Save Macro	563
END - End the Edit Session	564
ENUMWITH - Change Increment for Enumerate Functions	566
EOL - Set End-of-Line delimiter	567
EXCLUDE - Exclude Lines Edit Mode	567
EXCLUDE - Exclude Lines File Manager Mode	570
EXIT - Terminate SPFLite Session	571
FAVORITE - Add Current File To A Favorite List	572
FF - Find In Files	574
FIND - Find a Character String	576
FIND - Find File Name in File Manager List	579
FLIP - Reverse Exclusion Status of Lines	580
GLUEWITH - Specify Join String for GLUE operations	583
HELP - Display Online Help	584
HEX - Enable Hex Display of Data	586
HIDE - Hide Excluded or FILE Lines	587
HILITE - Control Text Highlighting Options	589
IMACRO - Set Initial Macro	591

INSTANCE - Switch to another Instance	593
JOIN - Join lines Using Find/Change Strings	595
KEYMAP - Display Keyboard Settings Dialog	601
LABEL - Manipulate Line Labels	603
LC - Lower Case a Range of Lines	605
LINE - Apply Line Command	607
LOCATE - Scroll to a Specific Line	609
LOOPCHECK - Turn loop detection ON or OFF	614
LRECL - Set record Length	615
MACLIB - Specify Unique Macro Folder	615
MAKELIST - Create FILELIST	617
MARK - Turn Mark ON or OFF	619
MD - Make Directory	620
MEDIT - Add a File To a Multi Edit Session	620
MINLEN - Set Minimum Record Length	621
MODE - Set various Editing Modes	622
NDELETE - Delete Lines Where String is Not Found	624
NEXCLUDE - Exclude Where String is Not Found	627
NFIND - Find Where String is Not Found	630
NFLIP - Negative Reverse Exclusion Status of Lines	633
NOTIFY - Set Temporary File Notification Level	636
NREVERT - Negative Revert of User Line to Ordinary Line	637
NSHOW - Show Lines Where String is Not Found	639
NULINE - Negative Mark of User lines	642
OPEN - Open New SPFLite Instance and Edit File	644
OPTIONS - Set Global Editor Options	645
PAGE - Set PAGE option	647
PASTE - Paste Data from the Clipboard	649
PREPEND - Insert text as start of line	652
PRESERVE - Control Handling of Trailing Blanks	654
PRINT - Send Selected Lines to the Printer	656
PROFILE - Display Current File Profile Variables	660
PTYPE - Enter PowerType Mode	663
RCHANGE - Repeat Change	665
RECALL - Recall a Favorite File List	666
RECFM - Set Record Format	668
REDO - Redo an UNDO Action	668
RELOAD - Reload Current Edit File	669
RENAME - Rename the Current Edit File	670
REPLACE - Replace a File	672
RESET - Reset	674
RETF - Retrieve Forward Direction	677
RETRIEVE - Retrieve Previous Commands	678
REVERT - Revert User Line to Ordinary Line	679
RFIND - Repeat Previous Find Command	681
RLOC - Repeat Previous Locate Command	682
RLOCFind - Repeat Previous Find or Locate Command	683
RUN - Execute the current Script	684
SAVE - Save Data and Continue Edit	686
SAVEALL - Save All Current Tabs	687

SAVEAS - Save as New Name and Switch to it	688
SC - Sentence-Case a Range of Lines	690
Scroll Commands - UP / DOWN / LEFT / RIGHT	691
SET - Set a Command Variable	693
SORT - Sort the Edit Data	698
SHOW - Show Lines Where a String is Found	702
SOURCE - Specify Character Encoding	705
SPLIT - Split Lines Using Find/Change Strings	707
START - Set Initial File Position Option	716
STATE - Control Saving of Edit State Information	718
SUBARG - Set Default SUBMIT Argument	720
SUBCMD - Set alternate command for SUBMIT	721
SUBMIT - Pass Lines to an External Command File	723
SWAP - Switch to a Selected File Tab	728
TABS - Turn Tabs On or Off	730
TAG - Alter Tag Status of a Range of Lines	731
TC - Title Case a Range of Lines	735
TOP - Scroll to Top of File	736
UC - Upper Case a Range of Lines	737
ULINE - Mark User lines	738
UNDO - Undo Changes	740
VIEW - View a File in Browse Mode	741
WDIR - Open Windows Explorer	743
XSUBMIT - Submit an external file	744
XFORM - Specify a File Transform Macro	745
XTABS - Control Incoming Tab Characters	747
SPFLite Macro Support	748
Macros Introduction	748
Macro Format and Structure	750
Locating Macros and Include files	753
Accessing Macro Command Line Operands	755
Macros for fun and profit	767
Working with Global Data	775
SPFLite Interface	778
Interface Structure	779
Globally Stored Data	791
Working with Line Colors	796
Working with the Clipboard	799
Debugging your Macro	800
Function Overview	805
Function Details	818
Add_Array	820
Add_Line	820
Delete_Gbl_Num	820
Delete_Gbl_Str	820
Drop_Handle\$	821
First_Handle\$	821
FMGet_AbbrevName\$	822
FMGet_Attr\$	822
FMGet_Backup_Versions	823

FMGet_Cmd\$	823
FMGet_CreationTime\$	823
FMGet_Extension\$	823
FMGet_FCount	823
FMGet_FileAttributes	824
FMGet_File_Class	824
FMGet_FileName\$	825
FMGet_FilePath\$	825
FMGet_FilePattern\$	825
FMGet_FileSize	826
FMGet_FileSize\$	826
FMGet_FListNames\$	826
FMGet_Folder_Class	826
FMGet_FType	827
FMGet_LastWriteTime\$	827
FMGet_Lines	827
FMGet_Mode	828
FMGet_Note\$	828
FMGet_Path\$	828
FMGet_PRPN\$	828
FMSet_Cmd	829
FMSet_Msg	829
Get_ANSI2SOURCE_Table\$	829
Get_Arg\$	830
Get_Arg_Count	830
Get_Arg_KW	830
Get_Arg_KWGroup\$	831
Get_Arg_LRef\$	831
Get_Arg_LRef_Count	832
Get_Arg_NumLit\$	832
Get_Arg_NumLit_Count	833
Get_Arg_Tag\$	833
Get_Arg_Tag_Count	833
Get_Arg_TextLit\$	834
Get_Arg_TextLit_Count	834
Get_BottomScrn_LPTr	834
Get_BNDS\$	835
Get_BNDSLIne\$	835
Get_Clr_Line\$	835
Get_Csr_Col	836
Get_Csr_LPTr	836
Get_Curr_Line\$	836
Get_Curr_Path\$	837
Get_Curr_Word\$	837
Get_Dest_LCmd\$	837
Get_Dest_Op	838
Get_Dest_LMod\$	838
Get_Dest1_LPTr	838
Get_Dest2_LPTr	839
Get_EnvVar\$	839

Get_EOL_STR\$	840
Get_EXE_Path\$	840
Get_FileBase\$	840
Get_FileDate\$	840
Get_FileName\$	841
Get_FileExt\$	841
Get_FilePath\$	841
Get_FileTime\$	841
Get_Find_First_Col	842
Get_Find_First_Len	842
Get_Find_First_LPTr	842
Get_Find_Last_Col	843
Get_Find_Last_Len	843
Get_Find_Last_LPTr	843
Get_First_LPTr	844
Get_FullPath\$	844
Get_Gbl_Num	844
Get_Gbl_Num_Count	845
Get_Gbl_Num_Name\$	845
Get_Gbl_Num_TableName\$	845
Get_Gbl_Str\$	846
Get_Gbl_Str_Count	846
Get_Gbl_Str_Name\$	846
Get_Gbl_Str_TableName\$	847
Get_Handle\$	847
Get_INI_Path\$	848
Get_Instances\$	848
Get_Label\$	848
Get_LBound	849
Get_Last_LPTr	849
Get_LeftScrn_Col	849
Get_Line\$	849
Get_Line_Len	850
Get_Line_Type\$	850
Get_LNum	850
Get_LOC_LPTr	851
Get_LPTr	851
Get_MacName\$	851
Get_MarkLine\$	852
Get_MaskLine\$	852
Get_Modified	852
Get_Modified_FileName	853
Get_Msg\$	853
Get_Next_LPTr	853
Get_Primary_Cmd\$	854
Get_Profile\$	854
Get_RBound	856
Get_RightScrn_Col	856
Get_RC	856
Get_Select_First_LPTr	857

Get_Select_Last_LPtr	857
Get_Select_Col	857
Get_Select_Len	857
Get_Session_Type\$	858
Get_SETVAR\$	858
Get_SOURCE2ANSI_Table\$	858
Get_Src_LCmd\$	859
Get_Src_Op	859
Get_Src_LMOD\$	860
Get_Src1_LPtr	860
Get_Src2_LPtr	860
Get_TabNum	861
Get_TabsLine\$	861
Get_Tag\$	861
Get_Trk_Pos\$	862
Get_Trk_LPtr	862
Get_Trk_TrkID	862
Get_TopScrn_LPtr	863
Get_Uniq_ID	863
Get_WORDCHAR\$	863
Get_XLINES	864
Get_XSTATUS\$	864
Halt	864
Has_Handle	865
Is_Available	865
Is_FM	866
Is_Line_Cmd	866
Is_Primary_Cmd	866
Is_xxxx	867
Last_Handle\$	867
Next_Handle\$	868
Prev_Handle\$	868
Reset_Gbl_Num	868
Reset_Gbl_Str	869
Set_Clr_Line	869
Set_Csr	869
Set_Gbl_Num	870
Set_Gbl_Str	870
Set_Line	870
Set_MARK	871
Set_MASK	871
Set_Msg	871
Set_SETVAR	872
Set_TABS	872
Set_TopScrn_LPtr	873
Set_WORD	873
SPF_Cmd	873
SPF_Debug	875
SPF_Exec	875
SPF_Exempt_File	876

SPF_FMLCmd	876
SPF_INS	877
SPF_InvChar\$	877
SPF_Loop_Check	878
SPF_OVR	878
SPF_OVR_REP	879
SPF_PARSE	880
SPF_Post_Do	881
SPF_Quote\$	881
SPF_REP	882
SPF_Shell	883
SPF_Trace	883
SPF_UnQuote\$	884
Working with The Interface	884
Working with Line Handles	895
Sample Macros	897
thinBasic Essentials	902
Overview	902
Data Types	904
Operators	906
Labels	907
Language Keywords	908
Flow Control	908
String Handling	912
Numeric Handling	926
User Interaction	931
Appendix	934
Introduction to Keyboard Primitives	934
Index to Keyboard Primitives	937
List of Keyboard Primitives	940
Supplied Fixed Fonts	955
Automatic Colorization Files	967
IBM ISPF Command Support	974
Abbreviations	978

Welcome to SPFLite

Contents of Article

[Introduction To SPFLite](#)
[SPFLite Evolution](#)
[Why Use SPFLite?](#)
[Version 1.0 thru 4.0](#)
[Version 5.0 thru 6.2](#)
[Version 7.0 thru 7.1](#)
[Version 8.0 thru 8.5](#)
[Version 9.0](#)
[Version 10.0 thru 10.2](#)
[Version 11.0](#)
[Release 2](#)
[Release 2.4](#)
[Release 2.5](#)
[Release 2.6](#)
[Release 2.7](#)
[Release 3.0](#)
[Release 3.1](#)
[A Note about Keyboard Functions](#)
[A Note about Screen Displays](#)
[A Note about System Compatibility](#)
[IBM ISPF Background Information](#)
[Licensing / Distribution](#)

Introduction To SPFLite

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. They were either too expensive and/or were poorly designed and **not** what I was looking for.

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.

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.

Along the way I was joined by Robert Hodge, who was, and is, an SPF aficionado. He has contributed many times to the code base, but mainly he has been the "Idea Man" and many of the extra features of SPFLite have come from his fertile mind. Together now for 15+ years, we have been refining and extending SPFLite.

Why Use SPFLite?

A lot of users might be thinking right now, "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

So, what's in it for you? *Lots*.

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 thru 8.5

Continuing addition of many new features: Updated File Manager, Dynamic colorization, command chaining, improved File Manager 'nesting', along with 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 thru 10.2

A total revision of screen color handling to integrate basic screen colors and those in colorization files. User control of Status Bar formatting, New commands DO and ALIGN. A new literal type - a delimited literal. This aids finding variable length items like strings inside brackets, braces, parenthesis etc. New support for Backup and Restore of files from

within SPFLite.

Version 11.0

This version provides several new features including additional Macro support:

- A Profile **IMACRO** option is provided to specify an Initial macro to be run immediately following loading of the file's data.
- Macro support has been expanded to allow use in the File Manager Tab. Additional macro functions specific to operating in the File Manager Tab have been added.
- A new "Handle" concept added for macro line identification.
- New Command line options were added to allow specification of an initial macro to use against the filename specified in the command line, and a separate option to allow specifying a DO macro to be run when initialization is complete.
- A new Keyboard primitive (**PA2**) is available. Like the PA2 key on the original 3270 terminals, it allows you to 'throw away' typing done on the current screen, so long as PA2 is entered **before** entering any other attention key (like *Enter*, *PgUp*, *PgDn*, *PFn* etc.)
- A new sort option for File Manager - **Name*** which requests the list be unsorted. i.e. in the sequence file information was retrieved.
- Improved File Manager History to allow moving Backward and Forward through your history of different File Manager views. Supported by two new primitive keys (**FMBack**) and (**FMFwd**)

Release 2

The changes in this 'version' were significant enough to warrant calling this **Release 2**.

The whole methodology of storing program parameters and preferences was altered. Support was added to provide flexibility in storing this data, as well as support for running multiple Instances of SPFLite, each with their own possibly unique preferences and parameter storage locations. See ["Managing SPFLite Instances and Configurations"](#).

Version 2.1 and 2.2 had many bug fixes, but also moved SPFLite2 out of the old folder to separate it from the older V11 modules. DIFF support was added for performing File comparisons.

Release 2.4

This release contained significant changes to the File Manager support. Additional data columns were made available, including a subset of the extended properties (MetaData). Many other small improvements to FM capabilities were made.

Release 2.5

This release was mainly a cleanup release. The primary command scanning and Find/Change control area were simplified and made more consistent. An extensive improvement to the HELP command was performed to provide a more useful search for appropriate help topics.

Release 2.6

This release introduced [Tracking](#), a new assist in moving around a document. As well, QUERY was replaced with standardized responses from option commands Numerous other 'tweaks' and enhancements.

Release 2.7

This release replaced the former Profile **USING** support with a new **Extended File Types** support. This provide a much more flexible method of identifying file types and assigning them to an actual Profile. Also enhanced was user control over the appearance of the Status Bar.

Release 3.0

This release, along with some minor changes, added support for Vertical ScrollBars on the right of the Edit and File Manager screens. The Extended File Type support was extended to consolidate the DEFAULT SHR list and the Non-Text file list.

Release 3.1

This release is also primarily a cleanup release. Over the years, many new functions and features have been added, and honestly, they were not done as properly as they should have. Expediency took hold and many 'kludges' and 'shortcuts' were done to add the new feature. We ended up with many layers of code, each of which independently were 'OK', but in total their 'clutter' made the code very difficult to work with.

So this release cleaned much of this up. The main command parser was re-written, the generalized Search routine revised and simplified and as a byproduct of testing all this, many unreported bugs in various commands have been corrected. Several command functions had outlived their usefulness and were removed.

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!

Small Physical Screens

If you are using a very small screen (like on a small Laptop) there are times when some SPFLite pop-up dialogues may not 'fit' on the screen, and the **Done** and **Cancel** buttons on the bottom are not visible. So how do you "get out"? The Done and Cancel buttons all have Keyboard Accelerators assigned, so you can just hit **D** or **C** to make your choice.

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 Open Source under the GPL3 license terms. All versions are **fully functional** and the source is available on the SPFLite Web Site.

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 public download sites, however the version available on those sites may not be the latest release, so we recommend you always use the above link.

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

- 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 / VIEW 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 or folders 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.
- A DIFF compare facility to compare two files and report differences.
- 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 be performed.
ADD	Add a text line
ALIGN	Align string on a column boundary
APPEND	Add text to the end of lines
AUTOBKUP	Control automatic backup creation
AUTOCAPS	Control auto-capitalization of language keywords
AUTONAME	Specify an alternate colorization control file for a Profile
AUTOSAVE	Control automatic file save defaults
BACKUP	Create Date/Time stamped backups
BOM	Turn UTF BOM writing ON and OFF
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
CLS	Clear or Close the SPFLite Debug Window
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
DCB	Specify the RECFM, LRECL and EOL settings together
DELETE	Delete selected lines
DIFF	Perform a DIFF file compare between two files
DIR	Display folder containing this file in File Manager
DO	Execute a keyboard command file
DOWN	Scroll downward in the data
EDIT	Open a file for editing
EFT	Edit the Extended File Type criteria
END	End the edit session
ENUMWITH	Change Enumerate increment value
EOL	Set End-of-Line characters. Deprecated - Use DCB .
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
FORMAT	Set file characteristics
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
IMACRO	Specify an initial macro to run after loading a file
KEYMAP	Display keyboard settings dialog
LABEL	Search for a line and add a Line Label
LC	Lower-case a range of lines
LEFT	Scroll leftward in the data
LOCATE	Scroll the display to a specified line
LOOPCHECK	Turn loop detection ON or OFF
LRECL	Specify logical record length. Deprecated - Use DCB .
MACLIB	Specify unique MACRO library for a Profile
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
MODE	Set/Switch the Mode (Edit, Browse, View) or Find/Change defaults of
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
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
OPEN	Edit another file in a new session
OPTIONS	Set editor global options
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
RCHANGE	Repeat change
RECALL	Recall (Open) a favorite File list
RECFM	Specify Record Format. Deprecated - Use DCB .
REDO	Redo (back out) a prior UNDO action
RELOAD	Reload current edit file

RENAME	Rename the current edit 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
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
UP	Scroll upward in the data
VIEW	View a file in read-only mode
WDIR	Open Windows Explorer for this file folder.
XFORM	Specify a Transform macro to handle file reading / writing
XSUBMIT	Submit an external file
XTABS	Control handling of incoming tabs

Summary of Line Commands

A	After destination
AA	After block
B	Before destination
BB	Before block
BNDS	Display BOUNDS line
C	Copy line(s)
CC	Copy a block
COLS	Display Columns line
D	Delete line(s)
DD	Delete a block
F	Display first lines in excluded region
G	Glue lines together
GG	Glue lines together in block
H	Here-destination lines
HH	Here-destination block
I	Insert temporary new line(s)
J	Join lines together
JJ	Join lines together in block
L	Display last lines in excluded region
LC	Lower-case lines
LCC	Lower-case lines in block
M	Move lines
MM	Move lines in block
MARK	Set column markers
MASK	Set the Insert line model
MD	Make data lines
MN	Make NOTE lines
N	Insert permanent new line(s)
NOTE / xNOTE	Insert NOTE or xNOTE lines
O	Overlay lines
OO	Overlay lines in block
OR	Overlay-Replace lines
ORR	Overlay-Replace lines in block
PL	Pad lines
PLL	Pad lines in block
R	Repeat lines
RR	Repeat lines in block
S	Show lines
SC	Sentence-case lines
SCC	Sentence-case lines in block
T	Select text lines
TT	Select text lines in block
TABS	Display Tabs line
TB	Text Break a line
TBB	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 To Installation](#)
[Installing Updated Versions](#)
[Uninstall Considerations](#)
[Initial Execution of SPFLite](#)
[Installation Wrap-up](#)

Introduction To Installation

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.

If the new version requires changes to your existing configuration data, the new version will contain special logic to perform the migration automatically.

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.

SPFLite configuration files can be stored anywhere in your file system that you prefer, even on a Network drive. There are two parts to the configuration. They may, but do not have to, share the same folder.

Common Configuration File

This file (SPFLite.CFG) will contain all modifiable SPFLite configuration parameters.

SPFLite Data Sub-Folders

These files consist of several different working folders. It is here that you will create and store Macros, AUTO colorization files, Clipboard permanent storage etc.

SPFLite will default both of these to your `\Users\You\Documents\SPFLite\` folder.

A new install will present you with the following pop-up to make some initial choices.

SPFLite Welcome

Welcome to SPFLite Version 2.2.20170

This appears to be your first execution of SPFLite2, please complete the options below. Once you select A/B or C, you will then be taken to the standard OPTIONS panel which allows further customization.

Please enter the folder for the SPFLite2 Common Configuration file.

D:\Documents\SPFLite\

Please enter the folder for the SPFLite2 Data Sub-Folders.

D:\Documents\SPFLite\

(The above locations can be altered later via the OPTIONS command)

Choose one of the following settings for ENTER and NEWLINE

(A)

 Enter key is ENTER, Backspace key is NEWLINE

(B)

 Enter key is ENTER, Ctrl-Enter key is NEWLINE

(C)

 Right Ctrl key is ENTER, Enter key is NEWLINE

(You may alter these later via the KEYMAP command)

Note: This window will close, and Initialization may take a minute.
Please be patient, this is one-time delay only.

At this point, you should review and set the two folder options, and then choose whichever of the three options presented that seems closest to your preference. You can alter all of these choices later to whatever you desire, using the [OPTIONS](#) and [KEYMAP](#) commands.

A common problem for new users is to make wrong assumptions about which keys are mapped to the ISPF equivalents of ENTER and NEWLINE. If you are new SPFLite user, you may wish to make a note of which choice (A / B or C) you made.

Regardless of which you choose, you can **always** change these later using the KEYMAP facility.

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 To Portable Install](#)

[Retaining Customization](#)

Introduction To Portable Install

SPFLite can be copied to removable media (such as a USB thumb-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.

- Copy the entire SPFLite program files folder from C:\Program Files (x86)\SPFLite3 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 Common Configuration file is 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 SPFLite.CFG **file** to the folder created above.
- Next copy all the **folders** to the \Config folder on the thumb drive.
- 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.

Differences between SPFLite and IBM ISPF

Contents of Article

[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 (very few) 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 if desired, 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.

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

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.

- 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

- **ACTION** to request periodic automatic SAVES
- **ADD** to insert a string as a new line
- **APPEND** to add a string to each line
- **AUTOCAPS** to control capitalization of language keywords
- **AUTONAME** to specify alternate colorization control file
- **BACKUP** to create local backup copies of a file
- **BOM** to control leaders on Unicode files
- **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
- **DIFF** to compare two files and report the differences
- **DIR** to open a file display of the folder containing the current file
- **ENUMWITH** specify increment to use with Enum functions
- **EXIT** to close all active tabs
- **FAVORITE** to add current file to a Favorite FILELIST group
- **FORMAT** to specify file characteristics
- **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
- **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 SYSOUT 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

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** (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 **-** 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

[Standard Windows Dialogs](#)
[Scrolling](#)
[Cursor Positioning](#)
[File Manager Context Menus](#)
[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.

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.

File Manager Context Menus

While working in File Manager, if you right-click on one of the line items, a small Context menu will appear with the most common Line Commands listed. Simply Left-click on one of the displayed items to invoke it. Note you must be Right-clicking on a non-blank character in the line or your click will be ignored.

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 if in FilePath mode.

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.

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.

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

[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:

SPFLite Welcome

Welcome to SPFLite Version 2.2.20170

This appears to be your first execution of SPFLite2, please complete the options below. Once you select A/B or C, you will then be taken to the standard OPTIONS panel which allows further customization.

Please enter the folder for the SPFLite2 Common Configuration file.

D:\Documents\SPFLite\

Please enter the folder for the SPFLite2 Data Sub-Folders.

D:\Documents\SPFLite\

(The above locations can be altered later via the OPTIONS command)

Choose one of the following settings for ENTER and NEWLINE

(A)

 Enter key is ENTER, Backspace key is NEWLINE

(B)

 Enter key is ENTER, Ctrl-Enter key is NEWLINE

(C)

 Right Ctrl key is ENTER, Enter key is NEWLINE

(You may alter these later via the KEYMAP command)

Note: This window will close, and Initialization may take a minute.
Please be patient, this is one-time delay only.

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 would 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.

Note: All these can of course be altered to whatever you wish using the [KEYMAP](#) facility.

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
- **S-F12** RETF

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 to End-of-line
- **PAUSE** PA2 (Cancel this screen input)
- **Right-Alt** KEYMAP
- **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
- **ScrLock** 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. But probably you would not have enough experience to make choices, so press **DONE** or **CANCEL** to exit the dialog.

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

SPFLite options are grouped into three main 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**.

The selection of what Profile to use to edit a particular file can be tailored to suit your needs using the Extended File Type (EFT) support. This is an entirely optional feature, you do not **NEED** to use it, but it simplifies Profile management enormously.

If you choose to NOT use EFT

SPFLite will select a Profile based solely on the file's extension. i.e. a .TXT file will use a TXT Profile, a .CSV file will use a CSV Profile etc. If a file has no extension it will use the common Profile **DEFAULT**.

The 1st time you edit a new file type, you will be prompted to create a new Profile. Several options are presented at this point.

This works fine, but you will probably find you have an ever growing proliferation of Profiles, many of which are probably 99% identical. This sometimes means, when you decide to change a Profile value to better suit your needs, that you may end up having to make the same change repeatedly through your collection of Profiles.

If you use EFT

EFT provides a powerful masking ability to allow you to select files based on their extension, folder location, naming pattern etc. and assign a Profile for the selected group. It allows specification of a Profile along with several minor temporary Profile overrides. e.g. Use the DEFAULT Profile along with a SOURCE EBCDIC override.

Please review [Extended File Types](#) for a full description of the EFT capabilities. It is well worth while and we're sure you will find EFT invaluable.

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>**. Individual Profile options can be displayed by simply entering the option name followed by a ? operand. e.g. **RECFM ?** 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. These options are described in the [SET](#) command. The current **SET** names involved are:

SET ALIAS.shortname = longname

SET AUTOFAV.filepath = named_favorites_name

SET TRACK.cmdname = Y | N

SET MACLIB.maclibname = full_macrolib_path

SET MACROMODE.macroname = BLOCK | SINGLE

SET PENDING.macroname = Y | N

Options - General

Index of General Options

[Audible BEEP on Errors](#)
[Visual BEEP on Errors](#)
[Delete to Recycle Bin](#)
[Use WORD as the default for FIND/CHANGE commands](#)
[Only English letters A-Z and a-z are considered alphabetic](#)
[Default RESET will Revert User Line Status](#)
[Open Help screen in Full Screen mode](#)
[Display MacName on Message line](#)
[Only 1 SPFLite running](#)
[Re-Open last file\(s\) at Start](#)
[Warn on modified View file](#)
[Display splash screen on startup](#)
[Warn on Open of Non-Text files](#)
[Cut should default to NEW](#)
[Maintain screen position after Line Cmds](#)
[Minimize SPFLite to the System Tray](#)
[BACKUP Retention Criteria](#)
[Minimum command length for Retrieve](#)
[Command Separator Char.](#)
[Default # cols to use for Data Shifts \(Min 1\)](#)
[Number of Columns/Lines per Auto-Scroll](#)
[Notify tabs on external file change](#)
[Line Commands Repeat Limit](#)
[Check for SPFLite Update Interval](#)
[Number of UNDO levels](#)
[Enter The Invalid characters for P'.' Picture Literals](#)
[Display Char. for un-mapped characters in the current Screen Font](#)
[Cancel and Done Buttons](#)

SPFLite Global Options (DEFAULT)

General FM Submit Screen KBD Status Schemes Hilights Config

☒ Audible BEEP on Errors
☒ Visual BEEP on Errors
☒ Delete to Recycle Bin
☐ Use WORD as default for FIND/CHANGE commands
☒ Only English A-Z and a-z are considered alphabeti
☐ Default RESET will Revert User line status
☐ Open HELP screens in Full Screen mode
☒ Display Macroname on Message line

☒ Only 1 SPFLite running
☒ Re-Open last file(s) at start
☒ Warn on modified View file
☐ Display splash screen on startup
☐ Warn on Open of Non-Text files
☐ CUT should default to NEW
☒ Maintain screen position after Line Cmds
☐ Minimize SPFLite to the System Tray

BACKUP Retention Criteria

RETPD [Days] Minimum Gens [Num] Maximum Gens [Num]

Minimum command length for RETRIEVE

Command separator character

Default # cols to use for data shifts (Min 1) (Append 'P' to use Profile XTAB)

Number of columns/lines per Auto-Scroll

Notify tabs on external file change


Line Commands repeat limit (0=no limit)

Check for SPFLite update interval

Number of UNDO levels (0 to turn off UNDO)

Enter the invalid characters for P'. ' picture literals

Display Character to use for Invalid characters (Or N to suppress translation)

 SPFLite.CFG file Folder:

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.

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 permanently deleted (i.e. deleted without recourse) bypassing the Recycle Bin.

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.

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

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.

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.

See further information on specifying the character values in ["Working with Word and delimiter characters"](#)

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.

Open Help screen in Full Screen mode

If checked, then when Help screen are displayed in response to a HELP command, they will

be opened in Full Screen mode. If unchecked, they will open in whatever size & position you have previously used.

Display MacName on Message line

There are times when some confusion may exist if you create macros that replace or 'front-end' normal SPFLite commands. You enter a normal SPFLite command and it fails, or acts differently. Hmmm, maybe you forgot that you ever created a macro of the same name.

If selected, then after any interaction that invokes a macro, the macroname will be displayed at the left end of the Message line to indicate a MACRO was invoked and not a normal SPFLite command.

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.

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 check-box 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.

Cut should default to NEW

When selected, all **CUT** commands will be treated as if the **NEW** operand were coded. i.e. if the cut is directed to a private named clipboard, then SPFLite will check to see if it exists. If it does, you will be prompted for permission to overwrite the existing file. If this option is selected, you may also use the **REPLACE** operand to override **NEW** and save without further prompting.

Maintain screen position after Line Cmds

When selected, SPFLite will attempt to keep the screen position as it was at the time of the

Attention (Enter or PFK). This effectively means that screen (positioning) commands issued by Line Commands outside the currently visible window will be ignored. This is not a 100% guarantee, depending on the various command combinations that are possible, but will usually prevent the screen from 'moving' away from it's last position.

Minimize SPFLite to the System Tray

If selected, when SPFLite is minimized, it will shrink to just an Icon in the System Tray (the area at the right end of the Windows Task Bar). It will be invisible in the normal Task Bar area showing active programs. Please ensure your Task Bar settings are altered to make SPFLite "Always Visible" in the System Tray area. Otherwise it could end up in the 'hidden' items and you will need to click on the ^ button to make it visible. If you use the [Instances](#) support, the ToolTip which appears when you hover over the Icon, will display the Instance Name if it applies to other than the DEFAULT session.

Display splash screen on startup

If selected, SPFLite will display an informational 'splash' screen at startup for a few seconds.

Warn on Open of Non-Text files

If selected, when a file is opened for Edit, SPFLite will check the filename to see if it is flagged as a Non-Text file. If so, it will prompt you for confirmation before continuing. Editing such files may cause corruption of the file so as to make it unusable.

Files are marked as non-text files using the EFT support. Review [Extended File Types](#) for details.

BACKUP Retention Criteria

If you use the Backup / Restore support built into SPFLite, this is where you specify the Retention Criteria which controls how long the Backup files are retained before they are considered 'obsolete' and can be deleted.

There are three values to be entered, a RETPD (Retention Period) value, a minimum Generations value and a maximum Generations value. These three values are all inter-related so before completing this section be sure you have read and understand how these values are used. It is **critical** you understand this or you could easily negate or cripple the whole Backup / Restore process. Review ["Working with Backup and Restore"](#)

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 Character

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 the IBM method.

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 of the rule above. You can use the ANSI popup in the KEYMAP to select some rarely used character that is convenient for you.

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.

Default # cols to use for Data Shifts (Min 1)

This setting specifies the default number of columns to be shifted when using one of the line shift commands { **(())**, **<<>>**, or **[]** } and no specific number 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.

Profile Override

It is possible to override this global default and use a value specific to each file Profile. This is done by appending a "P" (for Profile) to the numeric value. When this is done, the value used will be the XTABS (Import Tabs) value from the Profile in use. If the Profile has XTABS=0, then the numeric value from this global setting will be used.

Examples:

A value of **4** would mean - always use a value of **4**.

A value of **4P** would mean - use **4** if the Profile has XTABS = 0, otherwise use the Profile XTABS value.

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.

Check for SPFLite Update Interval

SPFLite can check periodically to see if a newer version has been made available on the SPFLite Web Site. This option allows you to choose one of three intervals:

Check Manually - SPFLite will do **no** automatic check. To do so you must manually click on the **Check Now** Button.

7 Days - SPFLite will check every 7 days.

30 Days - SPFLite will check every 30 days.

Check Now - Clicking this button will perform an immediate check. If your version is up to date, you will see:



If a Newer version exists, you will see:



If you choose, you may click on the website URL to go directly to the official download site.

Number of UNDO Levels

This setting controls whether SPFLite will support the **UNDO** command, and if enabled, the number of rollback levels of **UNDO** that are supported. If non-zero, it specifies how many levels of **UNDO** are supported. i.e. How far back you can go.

When the value is zero, **UNDO** is effectively OFF.

Enter The Invalid characters for P'. ' Picture Literals

The characters in this text box define what characters are considered "invalid" by the Picture character ' . ' (dot) when performing Picture searches.

Note that the dot Picture is used to identify characters which are considered "non-normal" characters. Normally this means characters which are not normally displayable on the screen, but you can use it to identify **whatever** characters **you** consider "non-normal", even if they are displayable.

That is, a Find Picture of ' . ' dot will find characters that **are** in the list.

The default list is based on the basic ASCII character set, you can customize it any way you see fit. If you erase the field, the default will be set.

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 in this list.

See further information on specifying the character values in ["Working with Word and delimiter characters"](#)

Display Char. for un-mapped characters in the current Screen Font

When SPFLite displays characters on the screen, and a character is encountered which is **NOT** defined in the current screen font, SPFLite will use the character you specify here in place of it. 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. You may specify any character (even a blank). If you enter a "N" or "n" it is taken as a request to do **NO** translation of invalid characters (not recommended).

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](#)

[Use Simple ANSI sort method](#)

[Display File Manager Help](#)

[Highlight Recent / Active dates](#)

[Enter the STANDARD column layout, in order](#)

[Enter the ALTERNATE column layout, in order](#)

[No. files in recent lists](#)

[Width of Command Line area \(Min 5\)](#)

[Initial FM column sort](#)

[Directory Placement](#)

SPFLite Global Options (DEFAULT)

General	FM	Submit	Screen	KBD	Status	Schemes	Hilights	Config
---------	----	--------	--------	-----	--------	---------	----------	--------

☐ Close SPFLite with last file tab

☐ Confirm file Deletes

☐ Use simple ANSI sort method

☐ Display File Manager Help

☒ Highlight Recent / Active dates

Enter the STANDARD column layout, in order

NAME(40),LWDATE,SIZELONG

Enter the ALTERNATE column layout, in order

NAME(40),PATH(40),LwDATETIME,SIZELONG

Valid column names to choose from are:

ATTR, CRDATE, CRDATETIME, LINES, LRDATE, LRDATETIME, LWDATE, LWDATETIME, NAME, PATH, PRP1, PRP2, PRP3, PRP4, PRP5, PRP6, SIZESHORT, SIZELONG, PRPn entries may request any of the following Properties, please read the Help file for full details.


ALBUM, ARTIST, BDEPTH, BRATE, CONTRIB, CAMMAKER, CAMMODEL, DATETAKEN, DIMENSION, DURATION, EXP, FLASH, FSTOP, GENRE, HRES, ISO, PUBLISHER, RATING, SUBTITLE, TITLE, TRK, VRES, YEAR

Number of entries in recent lists

35

Width of Line Command area (5 to 20)

6

 SPFLite.CFG file Folder: D:\Documents\SPFLite\

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 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.

The confirmation will tell you whether the file will be Recycled or will be 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 will proceed without this prompt. Because this helps you avoid deleting files by mistake, you are encouraged to enable this option.

Use Simple ANSI sort method

By default, SPFLite will sort FM file lists using the same sort sequencing as Microsoft Explorer. i.e. numeric fields within the filename are treated logically, not as ANSI text strings.

Example: The numbers 01, 02, 1 would be sorted as 01, 1, 02
If sorted by simple ANSI, they would be sorted as 01, 02, 1

If you select (Tick) this option, SPFLite will use the simple ANSI sort

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 un-checking 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 done using the following colors to mean:

RED	The file has been modified since 10 minutes before the beginning of this SPFLite session.
YELLOW	Modified between one hour before the start of this SPFLite session and the session start.
GREEN	Modified between eight hours before the start of this SPFLite session and one hour before the session start.
BLUE	Modified between twenty-four hours before the start of this SPFLite session and eight hours before the session start.

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

Enter the STANDARD column layout, in order
Enter the ALTERNATE column layout, in order

With the introduction of additional data columns, and also extended file properties in Release V2.4, SPFLite now provides two alternate layouts for use. For example, normal file lists would want to display common file attributes like Size, Last Write Date, Lines etc. But for files related to various media types (Images, Videos or Audio) the preference might be for data columns such as Artist, Title, Length etc. SPFLite provides two different Layouts, referred to as STANDARD and ALTERNATE for you to specify.

In these boxes, enter from left to right, the optional File Manager information columns you wish to be displayed, separated by commas
 Each column name can specify the desired column width by appending a value in parenthesis. e.g. **NAME(45)** The **PRPn** column names have an extended format (see below). If no column width is specified, an internal default will be provided.

The available columns are:

- **NAME**
This will contain the filename. (e.g. TEXTFILE.TXT)
- **PATH**
This will contain the path to the filename (e.g. C:\Users\Me\Documents\)
- **ATTR**
This column will have a short string of single character indicators representing the File Attributes. Typically this will only be 1-4 characters, however there are many *possible* attributes. The attributes and their codes are:

N = Normal	C = Compressed
D = Directory	P = Sparse_File
R = Readonly	U = Device
H = Hidden	L = Reparse_Point
S = System	O = Offline
A = Archive	I = Not_Content_Indexed
T = Temporary	E = Encrypted
	V = Virtual

- **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**
There are 3 different Dates available, each in two formats.
 - **LWDATE** is the date the file was **Last Written**.
 - **CRDATE** is the date the file was **CR**eated
 - **LRDATE** is the date the file was **Last Referenced**

Note: The Last Referenced date for actual Windows files is basically useless. The Last Referenced date is only valid for the SPFLite maintained Recent

Files/Paths lists.

Each of the 3 dates is available in the following formats.

LWDATE, CRDATE and LRDATE
MM-DD

- will display as YYYY-

LWDATETIME, CRDATETIME and LRDATETIME
MM-DD HH:MM

- will display as YYYY-

- **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.

- **PRPn**

There are 6 available Extended Attribute columns available (**PRP1** thru **PRP6**). These allow you to access some of the many MetaData fields available for some file types (Images, Audio, Video etc.) When specifying one of these, the normally optional (..) operand is now mandatory, and has been extended to contain 3 sub-operands. The full format for a **PRPn** entry is:

PRPv (nn, L/R, property-name)

Example:

PRP1 (9, R, Duration)

Where: **nn** is your desired width of the field to be displayed.

L/R specifies whether the data in the field is to be **Left** aligned, or **Right** aligned. All property data is returned as normal text strings, even if it contains numerical data (like Duration)

property-name is one of the supported extended properties (see below)

Note: The length must be at least as long as the *property-name* + 1, since SPFLite uses the column heading as the clickable area to request column sorting, and uses 1 additional character to insert the type of sort (+-*)

As well, the *property-name* is case sensitive. The column heading will be displayed exactly as you enter it here.

The selected columns will appear in the order you specify them. If more column data is requested than will fit in the screen width, the extra data will be created 'off-screen' The FM display can be scrolled Left/Right using the normal LEFT/RIGHT commands. The first column will be "frozen", the remaining columns will be scrolled.

Available property-names

ALBUM	Album name
ARTIST	Artist
BDEPTH	Bit Depth
BRATE	Bit Rate
CONTRIB	Contributors

CAMMAKER	Camera Maker
CAMMODEL	Camera Model
DATETAKEN	Original Date Taken - yyyy/mm/dd:hh:mm:ss.nnn
DIMENSION	Horizontal x Vertical dimensions
DURATION	Duration in mm:ss
EXP	Exposure
FLASH	Flash setting
FSTOP	F-Stop
GENRE	Genre
HRES	Horizontal resolution
ISO	ISO setting
PUBLISHER	Publisher
RATING	Un-Rated or 1 to 5 *
SUBTITLE	Sub-Title
TITLE	Title
TRK	Track number
VRES	Vertical resolution
YEAR	Year

No. files in recent lists

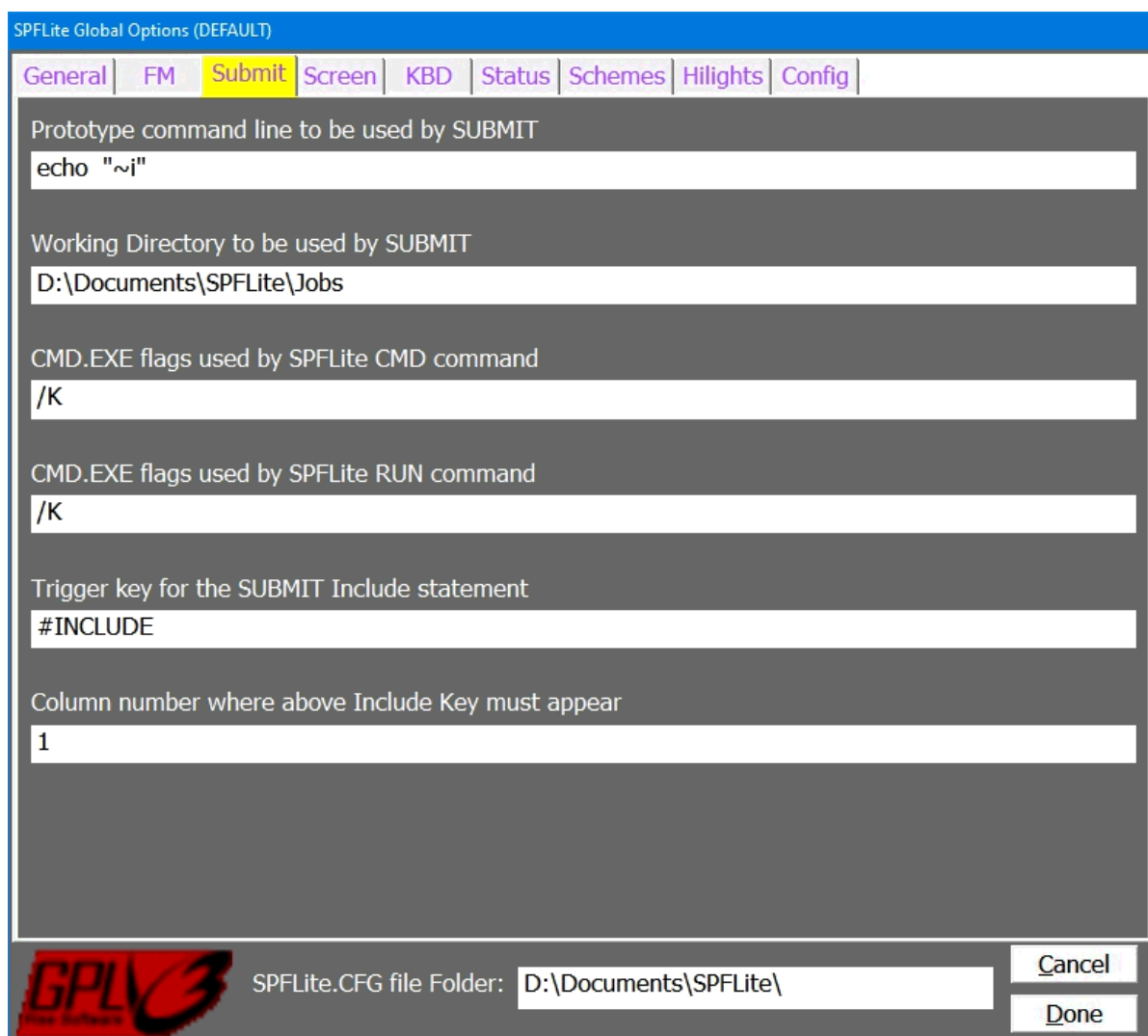
This setting specifies the maximum number of files to be retained in the **Recent Files** and **Recent Paths** file lists. The lists will automatically retain this number of entries, and remove older, less used entries from the list. The maximum value you can use is 99.

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.

Options - Submit



SPFLite Global Options (DEFAULT)

General | FM | **Submit** | Screen | KBD | Status | Schemes | Hilights | Config

Prototype command line to be used by SUBMIT
echo "%~|"

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

SPFLite.CFG file Folder: D:\Documents\SPFLite\

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](#)
[Hiligh Cmd line keywords](#)
[Add Vertical ScrollBars](#)
[=FILE> Line Format](#)
[Width of Line numbers \(5-8\)](#)
[Screen Colors - Click button to change](#)
[Choosing the file-modified status in the tab header](#)

SPFLite Global Options (DEFAULT)

General FM Submit **Screen** KBD Status Schemes Highlights Config

☒ Vertical cursor in Ins. Mode ☐ Horizontal cursor ☐ Vertical cursor

20 % Height Normal Cursor ☒ Alternating Background screen colors

100 % Height Insert Cursor ☒ Hiligh Command line keywords

RasterTTF Font Name RIGHT,.- =FILE> line format

16 Font Pitch 1.0 Status Bar/Tab Title Font scale factor

Choose (or Choose Font here) 0 # K-Board help lines to show

6 Width of Line Numbers (5-8)

☐ Column Marker Line Color

FG BG1 BG2

Line Number Hi-Intensity	BG2 Alternate			
Line Number Lo-Intensity	BG2 Alternate			
Active Tab Modified	BG2 Alternate			
Active Tab Unmodified	BG2 Alternate			
Inactive Tab Modified	BG2 Alternate			
Inactive Tab Unmodified	BG2 Alternate			
PFK Help Lines	BG2 Alternate			
Status Line	BG2 Alternate			
FM Quick Launch Bar	BG2 Alternate			
Messages / Errors	BG2 Alternate			
Diff Deleted lines	BG2 Alternate			
Diff Inserted Lines	BG2 Alternate			

GPLV3 SPFLite.CFG file Folder: D:\Documents\SPFLite\

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

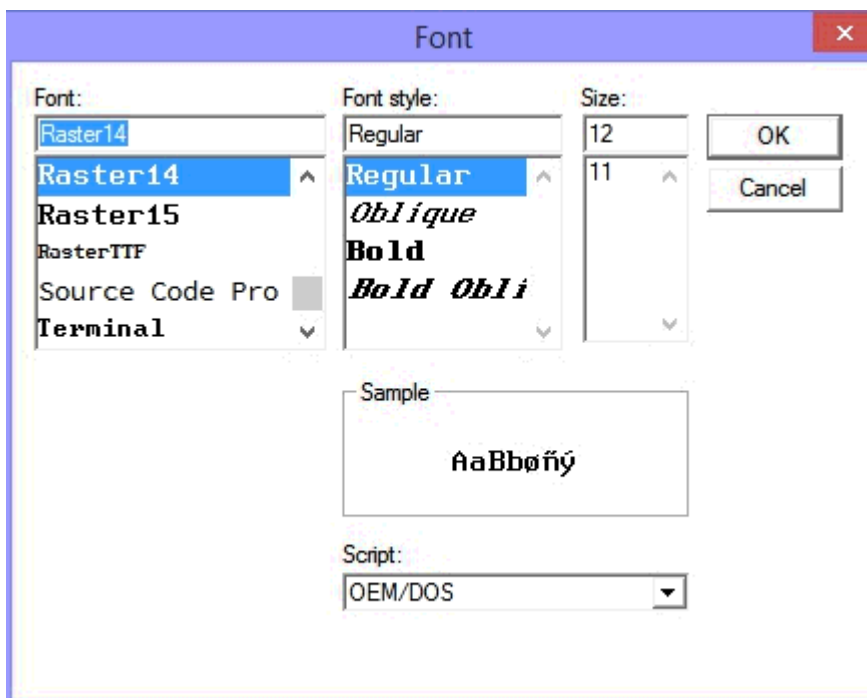
If checked, then a vertical bar cursor will be used when the keyboard is in Insert Mode rather than a square cursor (which can be adjusted by the next two items in this dialog). If this option is checked, the value chosen for **% Height Normal Cursor and Insert Cursor** will be ignored when in Insert 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. Raster15 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.

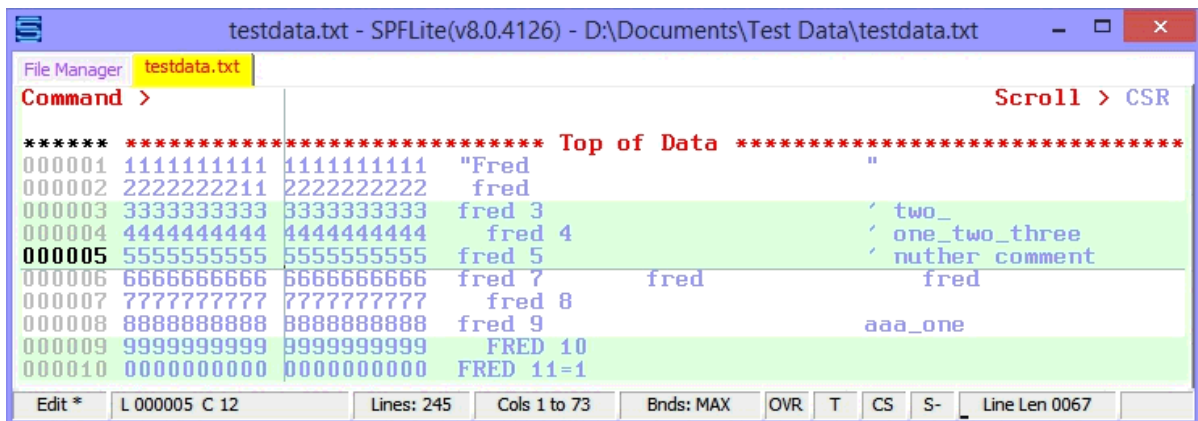
Column Marker Line Color

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 others 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. (See previous item) A sample of cursor rulers is shown below.



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 above and 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 FORMAT End-of-line AUTO 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

```

***** 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 fred
000007 7777777777 7777777777 fred 8
000008 8888888888 8888888888 fred 9 aaa_one
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-

Hilight 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"

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

File Manager testdata.txt

Command > change fred prefix bill all Scroll > CSR

```

***** 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 fred
000007 7777777777 7777777777 fred 8

```

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

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

Add Vertical ScrollBars

If you wish, you can request that vertical ScrollBars be added to the right side of the SPFLite screens. Some users find these useful, others prefer working without them.

To activate ScrollBars, simply select this item. SPFLite will need to be restarted to fully activate any change to this setting.

=FILE> Line Format

You may wish to have the =FILE> line (which marks the start of a File in MEdit mode) stand out or be more distinctive. This option allows you to describe how the line should look. The format of the entry is:

{ LEFT | RIGHT | CENTER | CENTRE } [,Pad-String] (The default is LEFT)

The first part indicates whether the Filename should be centered in the line, or Left/Right aligned.

The second operand is optional and indicates the padding of the rest of the line; the default is

- 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 12 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.

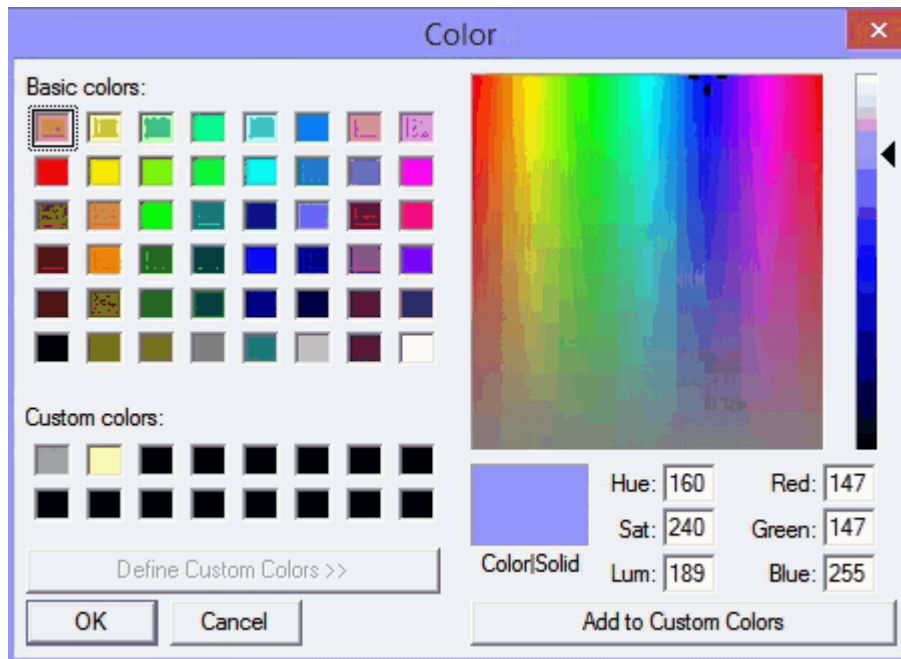
Messages / Errors Selection

The selection for Messages / Errors is slightly unique. Because Banding plays no part of the Message line display (line 2 on the screen) the BG1 and BG2 color selections are used to distinguish between simple informational messages, and error messages. The BG1 will be used for informational messages, the BG2 value for error messages.

Custom Colors

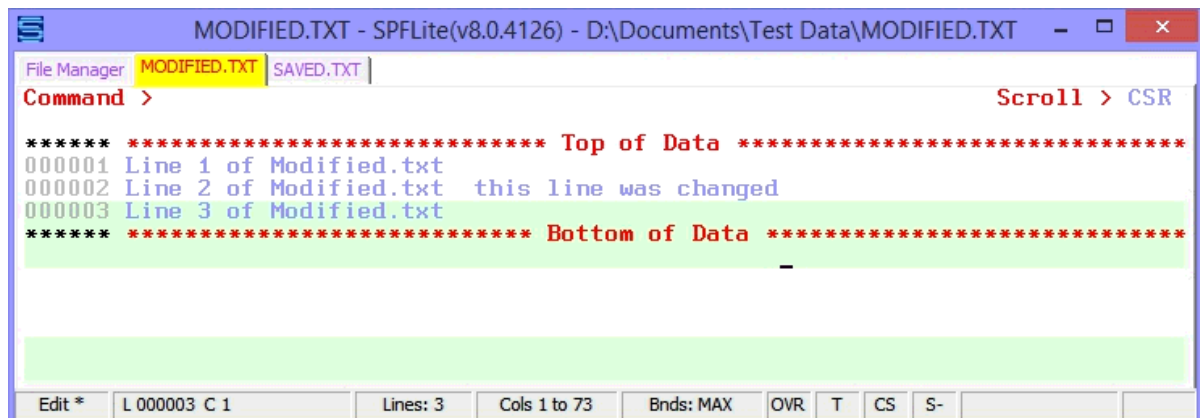
Note: When you create custom colors using the color chooser, SPFLite will "remember" these custom colors for your future use. This means you can create and save 16 custom colors.

Note: Before clicking "Add to Custom Color", if you want to select **which** of the 16 custom color boxes to save it to, Press TAB and then use the Arrow keys to select (highlight) the particular Custom Color box.

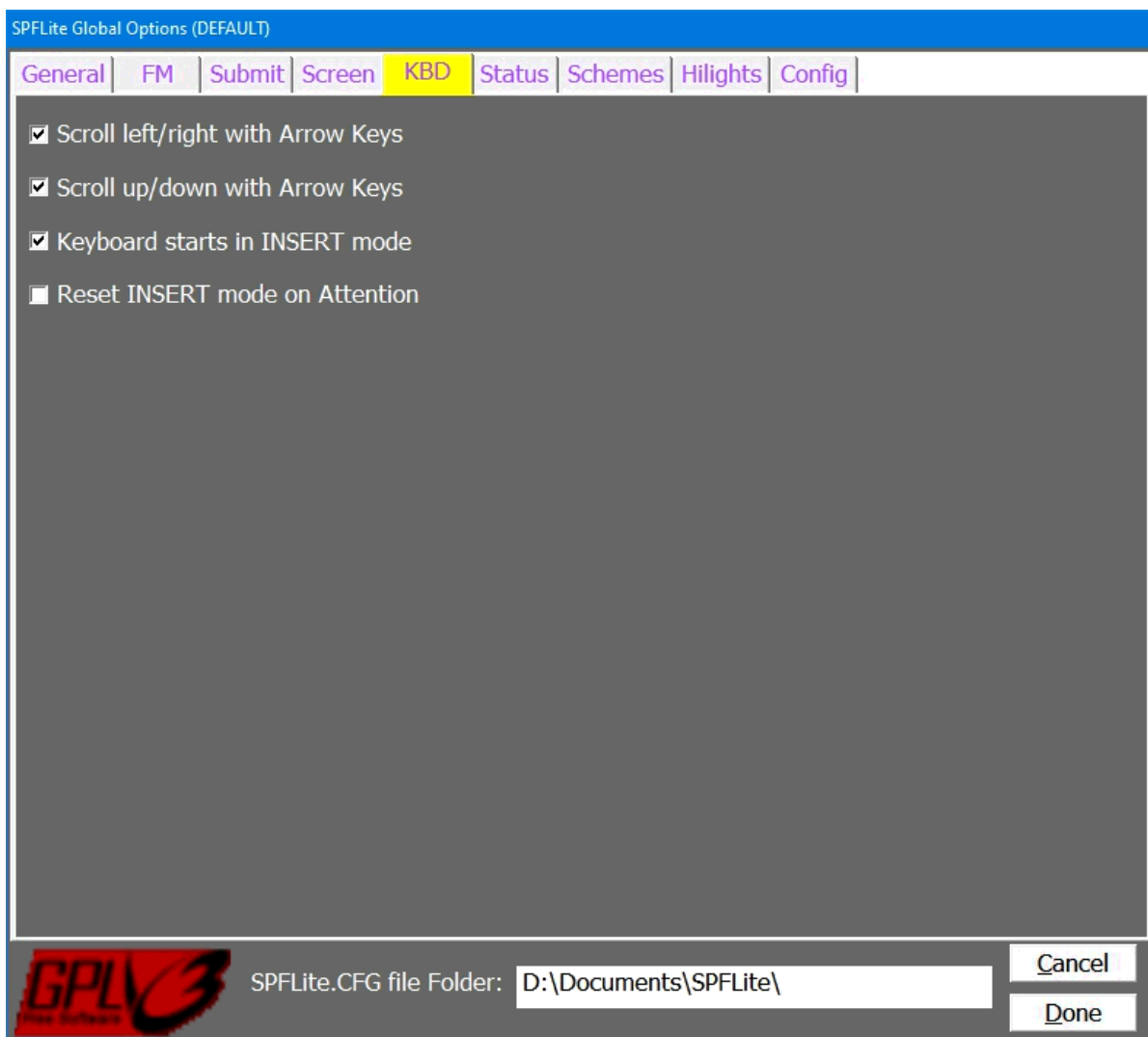


Choosing the file-modified status in the tab header

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



The screenshot shows the 'SPFLite Global Options (DEFAULT)' dialog box with the 'KBD' tab selected. The dialog has a blue title bar and a white tab bar with tabs for General, FM, Submit, Screen, KBD, Status, Schemes, Hilights, and Config. The KBD tab is highlighted in yellow. The main area is dark gray and contains four options, all of which are checked:

- ☒ Scroll left/right with Arrow Keys
- ☒ Scroll up/down with Arrow Keys
- ☒ Keyboard starts in INSERT mode
- ☐ Reset INSERT mode on Attention

At the bottom, there is a red 'GPLV3' logo on the left, a text field labeled 'SPFLite.CFG file Folder:' containing 'D:\Documents\SPFLite\' in the center, and 'Cancel' and 'Done' buttons on the right.

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.

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.

Created with the Personal Edition of HelpNDoc: [Streamline your documentation process with HelpNDoc's WinHelp HLP to CHM conversion feature](#)

Options - Status

SPFLite Global Options (DEFAULT)

General FM Submit Screen KBD **Status** Schemes Hilights Config

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(10),LINNO(15),LINES(10),COLS(15),BNDS(15),INSOVR(4),CASE(4),CHANGE(3),STATE(3),MISC(20),SELECT(10),SOURCE(8),EOL(7)

Valid field names to choose from are:

MODE(15),LINNO(20),LINES(15),COLS(23),BNDS(20),INSOVR(4),CASE(4),CHANGE(3),STATE(3),MISC(25),SELECT(30),CAPS(9),SOURCE(8),EOL(7)

Append (nn) to any field name to customize the width of the box
(nn) is the maximum message text, in characters; the value in the list above is the default.

Refer to Help => Working with SPFLite => Status Bar for more information

 SPFLite.CFG file Folder: D:\Documents\SPFLite\

Cancel Done

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.

Box Length Override

You can override the default width used for each box by appending a value, in parenthesis) immediately following the name. (As shown in the above screen shot). The value is specified in terms of the number of message text characters. Note this value is approximate as it calculates using an 'average' character width.

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

Note: The effect of the change will initially be seen only in the tab where the OPTIONS command was issued. It will take effect in other tabs as they are closed and re-opened, or at the next full recycle of SPFLite itself.

Created with the Personal Edition of HelpNDoc: [How to Protect Your PDFs with Encryption and Passwords](#)

Options - Schemes

SPFLite Global Options (DEFAULT)

General FM Submit Screen KBD Status **Schemes** Highlights Config

Scheme	Alias	Description	FG	BG1	BG2
	TxtLo	Text Low BG2 Alternative			
	TxtHi	Text High BG2 Alternative			
	Comments	Scheme 1 BG2 Alternative			
	Numbers	Scheme 2 BG2 Alternative			
	Quoted	Scheme 3 BG2 Alternative			
	Labels	Scheme 4 BG2 Alternative			
	Delims	Scheme 5 BG2 Alternative			
	Compiler	Scheme 6 BG2 Alternative			
	Data	Scheme 7 BG2 Alternative			
	LangKwds	Scheme 8 BG2 Alternative			
	LangFlow	Scheme 9 BG2 Alternative			
	APIEQU	Scheme 10 BG2 Alternative			
	APIFunc	Scheme 11 BG2 Alternative			
	Scheme12	Scheme 12 BG2 Alternative			
	Scheme13	Scheme 13 BG2 Alternative			
	Scheme14	Scheme 14 BG2 Alternative			

GPLV3 SPFLite.CFG file Folder:

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. (Text HI and Text LO Intensity)

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

Scheme Alias Names

To assist in 'remembering' your usage of each of the Schemes, you can alter the contents of

the Alias column to some name describing its use. Names are restricted to a single word, no spaces or commas allowed. The entries for the TxtLO and TxtHI values are fixed and unchangeable. Optionally, these Alias names can be used when creating your AUTO colorization files in place of hard-coded Scheme numbers. See [Automatic Colorization Files](#) for more info.

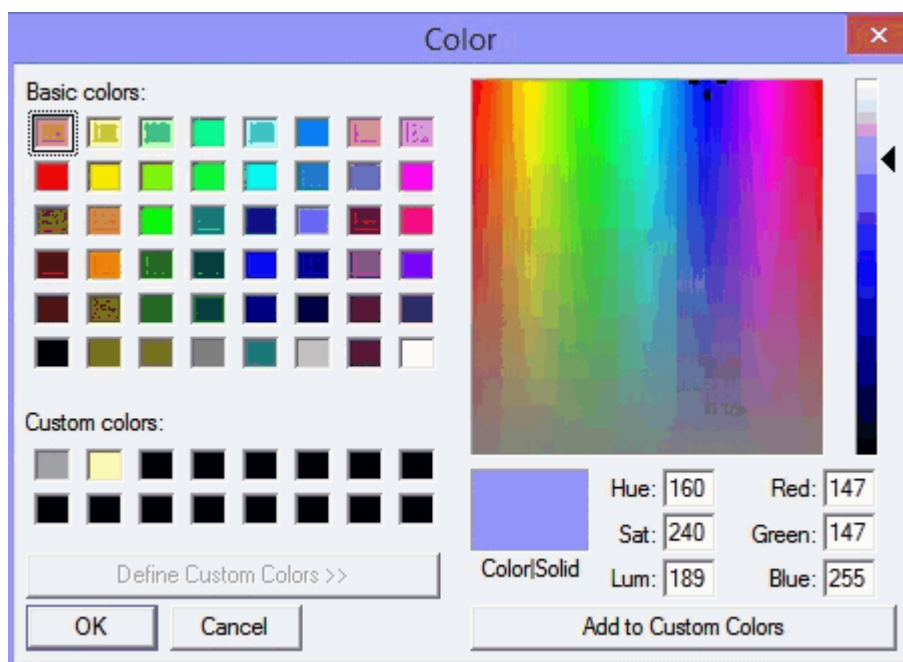
Color Selection

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.



Options - Highlights

SPFLite Global Options (DEFAULT)

General FM Submit Screen KBD Status Schemes **Highlights** Config

Highlight 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

SPFLite.CFG file Folder: Cancel Done

Specifying Hilight Color Selections

SPFLite allows you to hilight 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.

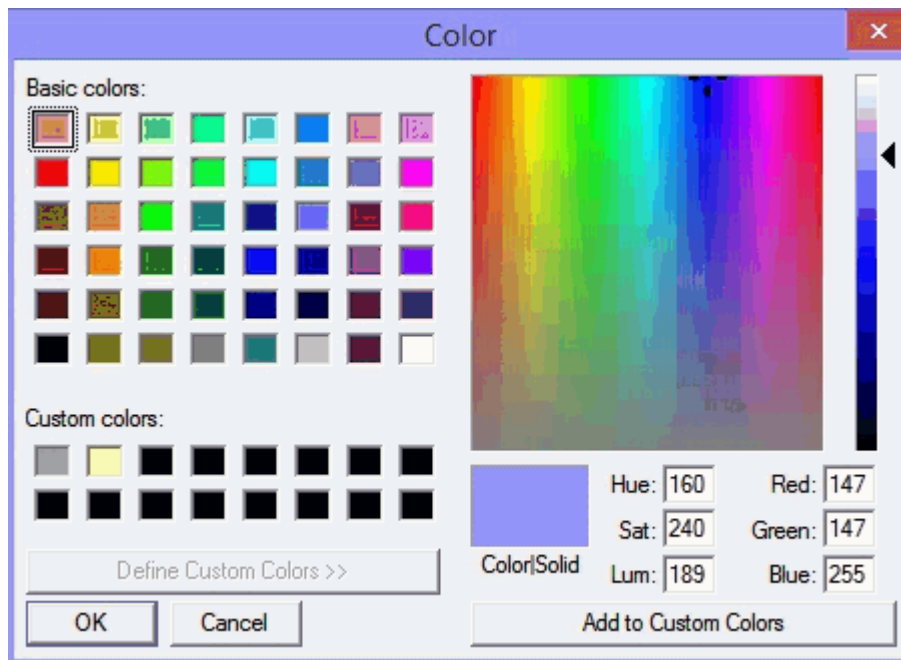
Each of the entries has an FG, BG1 and BG2 color selection. The FG specifies the color of the foreground text itself, the BG1 and BG2 specify the background color(s) Only BG1 is used if Banding is Off, both BG1 and BG2 are used if banding is On.

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: [Easily create Qt Help files](#)

Options - Config

SPFLite Global Options (DEFAULT)

GeneralFMSubmitScreenKBDStatusSchemesHighlightsConfig

The following Config values apply to the Instance: **DEFAULT**

This tab is for ADVANCED users only

If you are happy with SPFLite's previous handling of Options and the location of the SPFLite folder, or have not reviewed 'Working With Instances', leave these settings alone

The options below control the degree of sharing of configuration data
That is, some items can be shared with all Instances, and others can be maintained uniquely for this Instance

NOTES:

1. If You change an item from Shared to Unique, the initial unique settings will be copied from the DEFAULT Instance
2. A restart of SPFLite is generally needed to have these changes take effect
3. Changing Config folder locations requires the folder structure be built beforehand

☐ Select (Tick) if the EFT table is maintained uniquely


☐ Select (Tick) if the Keyboard table is maintained uniquely

☐ Select (Tick) if the Command Retrieve table is maintained uniquely

☐ Select (Tick) if the SET variable table is maintained uniquely

Location for the SPFLite Data storage folders (MACROS, AUTO, STATE etc.)

D:\Documents\SPFLite\

 SPFLite.CFG file Folder: D:\Documents\SPFLite\

CancelDone

Configuration Options

As the description at the top of the Config panel indicates, the settings on this page are for **advanced** users. If you have not read and reviewed [Managing SPFLite Instances and Configuration](#) then you should **NOT** modify the settings on this tab.

This tab allows you to modify the characteristics of an existing particular Instance (which is specified on the top line of the panel).

Note: For the DEFAULT Instance, only the locations of the Configuration files can be modified.

Select (Tick) if the EFT table is maintained uniquely.

This option controls whether the Instance will share the same Extended File Type table as the DEFAULT Profile, or whether it will use its own unique EFT table.

If changing from selected (Ticked) to non-selected, the existing unique keyboard table will be deleted and the Instance will now use the DEFAULT EFT table.

If changing from non-selected to selected (Ticked), the new, unique EFT table will be initially copied from the current DEFAULT EFT table.

Select (Tick) if the Keyboard table is maintained uniquely.

This option controls whether the Instance will share the same Keyboard definition as the DEFAULT Profile, or whether it will use its own unique keyboard table.

If changing from selected (Ticked) to non-selected, the existing unique keyboard table will be deleted and the Instance will now use the DEFAULT keyboard table.

If changing from non-selected to selected (Ticked), the new, unique keyboard table will be initially copied from the current DEFAULT keyboard table.

Select (Tick) if the Command Retrieve table is maintained uniquely.

This option controls whether the Instance will share the same Command Retrieve table as the DEFAULT Profile, or whether it will use its own unique retrieve table.

If changing from selected (Ticked) to non-selected, the existing unique Retrieve table will be deleted and the Instance will now use the DEFAULT Retrieve table.

If changing from non-selected to selected (Ticked), the new, unique Retrieve table will be initially copied from the current DEFAULT Retrieve table.

Select (Tick) if the SET variable table is maintained uniquely.

This option controls whether the Instance will share the same SET variable table as the DEFAULT Profile, or whether it will use its own unique SET variable table.

If changing from selected (Ticked) to non-selected, the existing unique SET variable table will be deleted and the Instance will now use the DEFAULT SET variable table.

If changing from non-selected to selected (Ticked), the new, unique SET Variable table will be initially copied from the current DEFAULT SET variable table.

Enter the location of the SPFLite Data Storage folder.

The Data Storage folder is the folder in which SPFLite will store and look for Macros, AUTO colorize files, saved CLIP files etc. Before changing this, read [Managing SPFLite Instances and Configuration](#) and ensure the new folder structure is 'in place' before making this change.

Configuration Folder is::

The Configuration folder is the location of the SPFLite Common Configuration file (SPFLite.CFG). Before changing this, read [Managing SPFLite Instances and Configuration](#) and ensure the folder and its content are 'in place' before making this change.

Created with the Personal Edition of HelpNDoc: [Make Your PDFs More Secure with Encryption and Password Protection](#)

Keyboard Customization and Keyboard Macros

Contents of Article

[Key Mapping does not apply to Windows dialogs](#)
[Nothing can go wrong, go wrong](#)
[Start SPFLite directly into KEYMAP](#)
[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 [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 itself
- The File Rename and File Delete 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 SPFLite
- 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.

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 of the SPFLite CFG file.

See [CFG File Maintenance](#) for details of how to Export and Import copies of your CFG 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 import one of your saved CFG export files back to the production CFG file, 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

[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 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 Ñ 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 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.

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 ñ not sign (X'AC') by holding down the ALT key and typing 0172, where X'AC' = decimal 172.

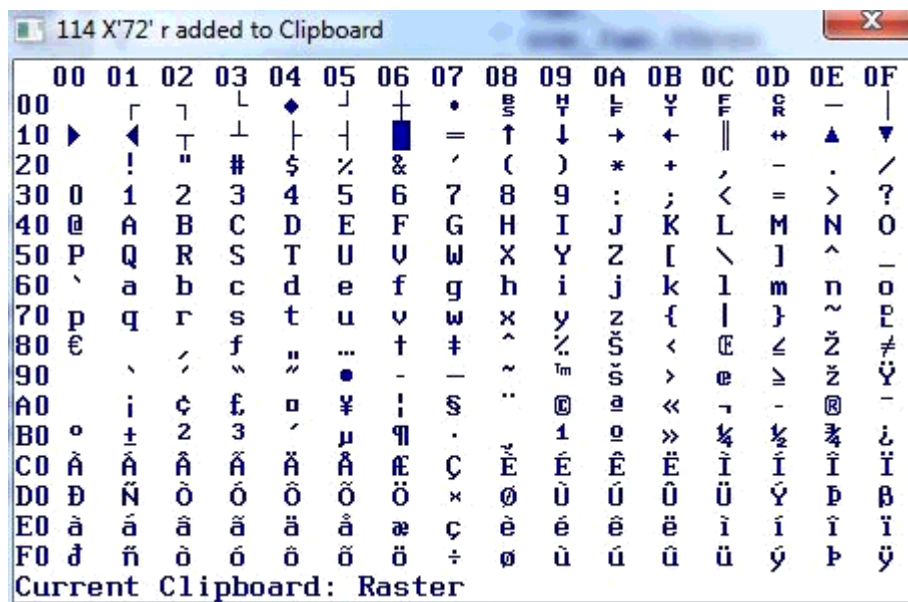
You have to actually enter the leading 0, or otherwise you get an OEM key value instead of an Ansi value, and for keys that are >= X'80' they won't be right. SPFLite uses Ansi encoding, not OEM encoding, and so 172 and 0172 are **not** the same character.

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

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 display 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 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.

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.

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.

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

Using the KEYMAP Dialog

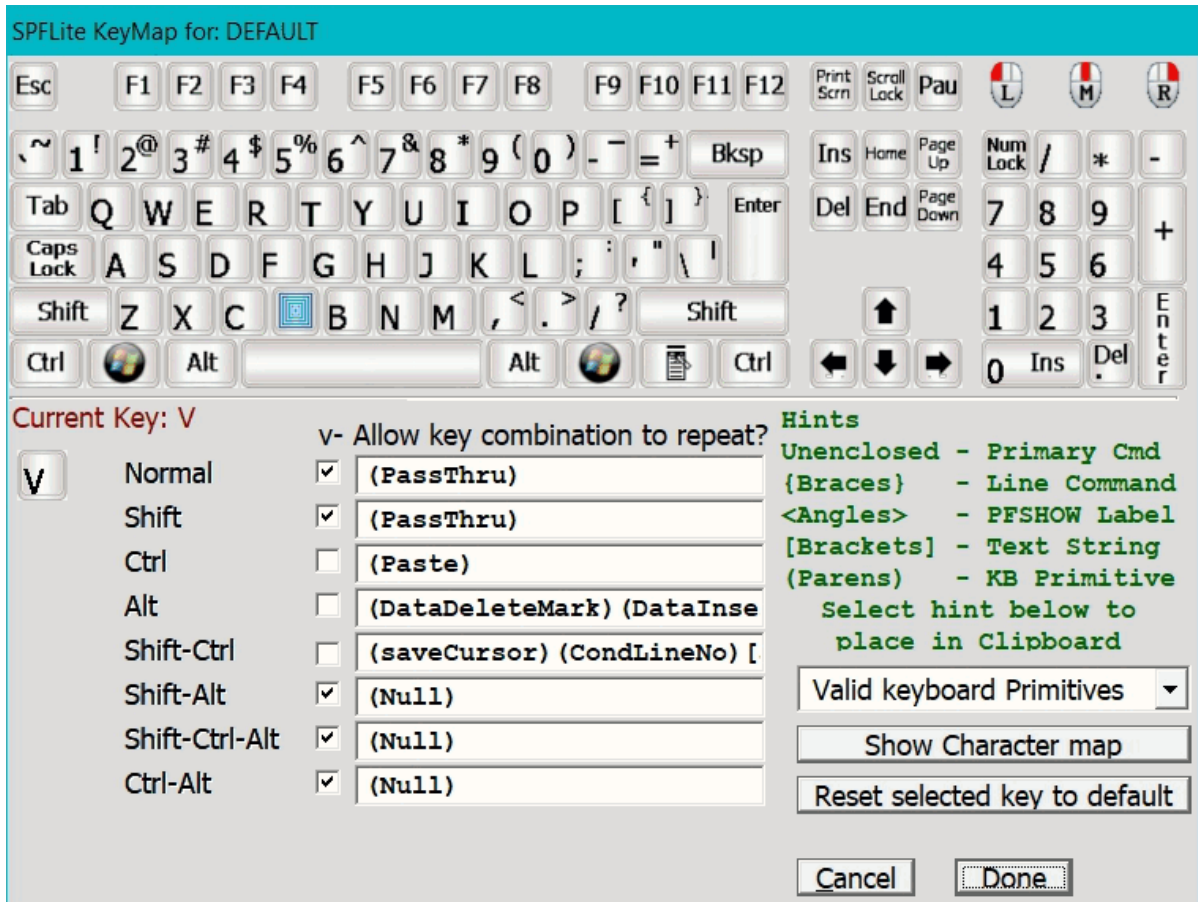
Contents of Article

[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](#)
[Auto Repeat / Set Cursor Position](#)

[Controlling the Keyboard Repeat Rate](#)
[Reset Selected Key to Default](#)
[Finalizing Your Keymap Customization](#)

Introduction

The SPFLite Key Mapping Dialog appears as follows:



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.

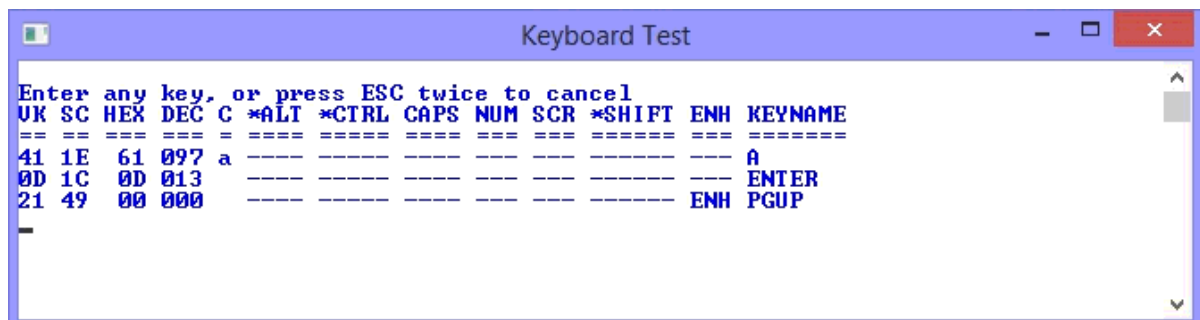
Non-US 104 Key Keyboards

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 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.

Note: If all you need to do is substitute a new single character for the key, simply enter that key character, there is no need to surround it with [] characters.

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.

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. The full description of available primitive keyboard functions can be found in the Appendix sections: [Introduction to Keyboard Primitives](#), [Index to Keyboard Primitives](#) and [List of Keyboard Primitives](#).

You can also paste any text into a definition field if 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.

Using the **!** command prefix is advised for all key maps which specify a 'complete' command. A 'complete' command is one where you do not wish/expect the key action to *preface* something you've already entered in

the Command line.

For example:

The 'complete' command **UP MAX** assigned to a key will fail if the command line is not blank when you press the key.

Using **!UP MAX** avoids this error because the **!** character causes the command line to be cleared before inserting the UP MAX command.

Exceptions to this conventions are key maps which deliberately allow one or more operands to be specified by the user.

For example:

UP, DOWN, LEFT and RIGHT

Allows the user to temporarily override the usual scroll amount by typing a number (of lines) or the letter M (max) in the command line before pressing the mapped key.

HELP

Allows direct access to help information about the command currently on the command line

RFIND

Allows the mapped key to find the first occurrence of the *<from>* string in a CHANGE command which is already present on the command line,

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

count)

- **(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].

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-*

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 *t* format, and allowing you to define long strings that would be awkward and *r* error-prone to type manually, this method is also significantly faster in *j* performance. This could be used, for example, to enter a long string of equal *n* characters.

^g
] 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.

Allow key to Repeat / Position Cursor on Mouse-Click

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.

Reset Selected Key to Default

This button on the dialog allows you to reset a key definition to the value SPFLite would have assigned on an initial installation. To use this, first select a key by clicking on the keyboard layout. Second, click in the text box for the specific key combination you want to reset (Normal, Shift, Control, Shift-Control etc.). This will place the cursor in the selected textbox. Then simply click on the *Reset Selected Key to Default* button.

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

[Creating a macro using the Record function](#)
[Keyboard recording and Power Typing](#)
[Saving/Editing key content 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 Keymap 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)**, a second time, recording stops. At this point complete any other PT activities and press Enter to leave PT Mode. Now you can enter KEYMAP to save the recording to your chosen key.

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 .z1                ; Cut all lines
clip
; Now in the Clip tab, change the data
; Overtyping 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 *labels*, 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**

Another method that supports 10 different bookmarks and allows direct 'jumps' to each, rather than doing 'next', 'next' etc.

It uses the numeric keys **1** thru **0** and the line labels **.BM_A** thru **.BM_J**

Set up the **Alt-1** key to **(SaveCursor){.BM_A}(Enter)(RestoreCursor)** and the **Ctl-1** key to **!L .BM_A**. Similarly, the 2 key using **.BM_B**; the 3 key using **.BM_C** etc.

Now, Entering **Alt-1** sets the **.BM_A** bookmark, and **Ctl-1** will return you to that bookmark. Each numeric key thus has an ALT to set the BM, and a CTL to return to that bookmark. Normally 10 unique bookmarks are more than enough to keep you happy.

Created with the Personal Edition of HelpNDoc: [Transform Your Word Document into a Professional eBook with HelpNDoc](#)

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](#)
[Specifying an Overriding Initial Macro to use](#)
[Specifying an Overriding XFORM Macro 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.

Note: The files will be opened in simple Edit, View or Browse Mode depending on their state at the previous shutdown.

If you have not enabled "Re-Open last file(s) at Start", or there were no open files active when last shutdown, 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 presented as the first tab, and the edit file tab as the second one.
- Drag-and-Drop a file onto an existing SPFLite window. The dropped file will be opened in a new file tab.
- Drag-and-Drop a Folder onto an existing SPFLite window. The folder will be opened in the File Manager 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.

When any of the optional **-XXXXX** operands are coded, the path/filename operand should always be entered **last**. If the filename contains spaces, the operand should be enclosed in quotes. e.g. "My Special File"

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.

-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 filename may be coded 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.

-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.

-DO macroname

If **'-DO'** appears as the command line, SPFLite will invoke the specified [DO](#) macro following all normal initialization functions. The SPFLite session will be switched to the File Manager Tab so the [DO](#) macro must be crafted to operate in that environment.

-FILEOPEN option

The **-FILEOPEN** option specifies the name of a FileOpen file containing the names of files you wish to have opened immediately after startup. The format of this file is described in ["FileOpen Startup Files"](#) . If the filename is entered as a simple `"*"`, it requests use of the standard `_FILEOPEN.TXT` file created by the [INSTANCE](#) command. If the filename is not a complete path, the SPFLite HomeFolder will be added.

-INSTANCE instance-name

This option allows you to run unique instances of SPFLite, each with its own customization options. SPFLite will display the *instance-name* in the window title, replacing the normal SPFLite version number. Please read [Managing SPFLite Instances and Configuration](#) for more details.

-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.

-NOLOOP option

If **'-NOLOOP'** appears as the command line, SPFLite will **NOT** activate it's normal loop detection logic. This detection logic is designed 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.

-SCRSIZE option

If **'-SCRSIZE'** appears on the command line, SPFLite override the normal screen opening, which re-opens the screen at the same size and position as of the last termination.

'option' may be either:

FULL - will open the screen in full-screen mode.

hhXww - where **hh** is the screen height (in lines), **'X'** is a constant, and **ww** is the screen width, in columns. e.g. **-SCRSIZE 24X80** requests a 24 line screen, 80 columns wide.

-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 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.

-WINE option

This option is no longer needed, SPFLite will automatically detect if it is running under WINE.

Example: The command line:

```
SPFLITE.EXE -I CUSTOM -B MyTestFile.txt
```

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

Specifying an Overriding 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 .NewProfile "My File To Browse.txt"

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

Specifying an Overriding Initial Macro to use

When you specify a filename on the command-line to be opened it is possible to specify an Initial macro to be performed as soon as the file is loaded. If the file is using a Profile which itself contains an **IMACRO** request, the command line macro is used instead of the Profile **IMACRO**.

Simply provide the macro name, preceded by a %. (percent sign) on the command line following the filename. If the filename was enclosed in quotes, the Macro name follows this trailing quote, and is not itself quoted.

e.g. SPFLite -E %MyMacro "MyFile.txt"

The above shows a request to Edit a file using the Initial Macro 'MyMacro'.

Specifying an Overriding XFORM macro to use

When you specify a filename on the command-line to be opened it is possible to specify an Initial **XFORM** macro to be used to perform the read and write functions for the file. If the file is using a Profile which itself contains an XFORM request, the command line macro is used instead of the Profile **XMACRO**.

Simply provide the macro name, preceded by a/. (forward slash) on the command line following the filename. If the filename was enclosed in quotes, the Macro name follows this trailing quote, and is not itself quoted.

e.g. SPFLite -E /MyXFormMacro "MyFile.txt"

The above shows a request to Edit a file using the Initial Macro 'MyXFormMacro'.

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, the **EXIT** command, the **=X** command and the **[X]** button on the right side of the Title Bar all perform the same exact function, shut down the entire SPFLite session.

When shutdown is requested, 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.

Note re: XFORM sessions

When SPFLite saves the list of files to be re-opened at the next startup, it will never include the files from any tabs loaded via **XFORM** support. **XFORM** support is unique and should only be invoked manually, to protect the integrity of the files. More details on XFORM support can be found in ["Working with XFORM"](#).

Basic Edit Functions

Contents of Article

[Direct Modification of Text](#)
[Oops - Back-out](#)
[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 / Overtyping 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 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.

You may sometimes see "Overwrite mode" described as "Overtyping 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](#) for more information.

Oops - Back-out

There are many times where you may be typing away and then realize that the cursor was not where you thought it was. e.g. you thought it was on the command line, and instead it was down in the middle of your actual text.

You've just 'clobbered' a whole bunch of text data and may not even be clear what was there so you can put it back. What to do? !

Just hit the **PA2** key (defaulted at initial install time to **Alt-1**). The screen will be re-displayed

as it was at the last screen refresh. All typing you performed since the last screen display will be 'thrown away'.

But you must hit PA2 BEFORE you press any other key that triggers an Attention.
Such as *Enter*, *PgUp*, *PgDn*, or any *PFn* key.

If you repeatedly press **PA2** before triggering an Attention with some other key, the screen display will alternate between the modified and unmodified versions of the screen. Simply leave the screen in your desired choice and proceed (with any Attention key, like **Enter**).

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 [OPTIONS -> General](#) 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](#).

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 PA2 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.

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.filemask = filelist-name
```

Here, **filemask** 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.

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".

BACKUP & RESTORE

Contents of Article

[Where are they kept?](#)
[What format are they in?](#)
[Managing Backup Files](#)
[Types of Retention Control](#)
[When are Retention Criteria Evaluated?](#)
[Requesting a Backup](#)
[Requesting a Restore](#)
[Backup File Format](#)

Introduction

SPFLite provides a simple to use file Backup and Restore facility. This enables you to easily create multiple date and time stamped backups for your files, on demand, and to restore them simply if and when needed. All without leaving the SPFLite session to run other tools.

Where are they kept?

In order to provide a consistent location, and to prevent 'cluttering up' your normal file storage folders, backup files are stored in a sub-folder of the normal file's location. SPFLite will create a new folder **\$BACKUP** in the file's normal data folder, and store all backups there.

Note: This means there is not one single **\$BACKUP** folder for your system, it is one **\$BACKUP** folder for each data folder you use for files edited by SPFLite. This simplifies Restore as the destination for the Restore is always the folder above the one which contains the actual Backup file.

Note: For Backups of special sessions like CLIP, DIFF, SetEdit and EFTEdit, the **\$BACKUP** folder used will be located in the normal SPFLite data storage folder (normally \USERS\you\SPFLite\)

What format are they in?

Backup files are 100% unmodified clones of the original file. Their only difference is the filename which has had a Date / Time-stamp portion appended. In addition, the file type extension is also repeated at the end. This means the file is perfectly usable, if needed, by its normal application without any additional processing. Backups of CLIP, SET and EFT sessions are created as normal TXT files.

e.g. For a file name of **TESTDATA.TXT**

The Backup file would be: **TESTDATA.TXT.190503-132132.190504.TXT**

where the time-tamp (**190503-132132**) is the Last Update date and time of the original file
and the next portion (**190504**) is the date the backup itself was created.

When the file being backed up has been edited by SPFLite and has an associated STATE file,

the STATE file will also be backed up and will take the same name as the original file's backup with **.STATE** appended. Using the example above it would be called:

TESTDATA.TXT.190503-132132.190504.TXT.STATE

Managing Backup Files

SPFLite provides you with the ability to request **NO** management support for your Backup files (i.e. you will perform this yourself outside of SPFLite) or to request SPFLite assist in managing your Backups under Retention criteria which you specify.

Regardless of what method you choose, SPFLite will always, when a BACKUP request is performed, inform you of how many current Backups there are for the file just processed.

Types of Retention Control

SPFLite allows you to specify Retention Control Criteria via the [OPTIONS => General](#) setting tab. This entry requires the entry of the three required fields. The RETPD (Retention Period), The Minimum Generations (MinGen) and the Maximum Generations (MaxGen). These three fields are all inter-related to control how long Backups are retained.

SPFLite supports the following types of retention.

Full User Control

This is indicated by setting all three values to Zero
(RETPD = 0, MinGens= 0, MaxGens=0)

When this is specified, SPFLite will **never** delete any Backups created, responsibility for Backup file cleanup lies with the user, using any means they desire.

Retention Period Only

This is indicated by entering in the RETPD the number of days you wish to retain the Backups and leaving the MinGens and MaxGens values as Zero.
(RETPD = **nnn**, MinGens=0, MaxGens=0)

When this is specified, the **nnn** value indicates the number of **days** each Backup file is to be retained before it becomes 'obsolete' and is deleted.

Generation Control

This is indicated by entering Zero for RETPD and MinGens and specifying the number of Backup generations you wish to keep as the MaxGens value.
(RETPD = 0, MinGens=0, MaxGens=**xxx**)

When specified in this format, the **xxx** value indicates the number of **generations** of each Backup file that are to be retained. Older generations over this value are considered 'obsolete' and will be deleted.

Generation / Retention Hybrid Control

This is indicated by entering values for all three control values.
(RETPD = **rrr**, MinGens=**mmm**, MaxGens=**xxx**)

When specified in this format, the **rrr** value indicates the number of **days** each Backup file is to be retained before it becomes eligible for removal and deletion. The **mmm** indicates the **minimum** number of generations of each Backup file that are to be retained. The **xxx** indicates the **maximum** number of generations of each Backup file

that are to be retained.

Wow! That's complicated! So how does this all work? It seems very confusing. It's really not too bad, here's the process of evaluating the backups for a single file:

- First, any Backups exceeding the maximum **xxx** value will be deleted, regardless of their expiry date.
- Second, all remaining backups **over** the minimum **mmm** value will be evaluated based on their Retention **rrr** value and will be deleted if over the number of days.

Thus, you will always have a minimum of **mmm** backups available, and a maximum of **xxx** backups available. The **rrr** value is basically used to keep the total number of backups as low as possible (between the **mmm** value and the **xxx** value).

When retention processing removes all files from a **\$BACKUP** folder, the actual **\$BACKUP** folder will itself be removed.

When are Retention Criteria Evaluated?

The criteria you specify for Backup Retention are evaluated at the following times:

At SPFLite Startup (Once per day)

SPFLite will, on its first startup of a day, perform evaluation of all known **\$BACKUP** folders to determine if any Backup files are eligible for deletion. This evaluation will be done in the background and will not impact the operation of normal startup.

At a specific Backup request (Including AUTOBKUP requests)

Whenever an actual backup is performed, SPFLite will evaluate the single **\$BACKUP** folder to which this backup is directed.

Note: When generation control is in effect, you may notice that **1** more generation might be retained than the Retention MaxGen value. This is normal as the evaluation is performed **before** the actual backup is performed. The additional generation will be removed at the next evaluation (probably the following day on startup). This condition is perfectly normal.

Requesting a Backup

There are several ways to request a new Backup be created.

1. In File Manager, enter **BACKUP | BACK | BK** as a line command next to the desired file.
2. While in Edit / View / Browse of a file, enter **BACKUP | BACK | BK** on the command line.
3. Turn **AUTOBKUP ON** for the profile used by the file. AUTOBKUP will trigger a Backup for a file:
 - At most ONCE per Edit session. i.e. a Backup of how the file looked when Edit started.
 - Only if a SAVE is performed (or the file is Saved during END processing)
 - Only for the first SAVE issued. (Variation of the comment in the 1st bullet above)

For ALL cases, a backup will not be created if one already exists for the file and the Last Modified Date & Time stamps are the same.

Note:

If SPFLite detects the presence of an existing backup of the file, with the same Date/Time stamp, it will not create a new backup and will also verify the file size in addition to the date time.

If it rejects the backup and indicates the files are the same size, all is well. (Simply - no Backup is needed)

If it rejects the backup and indicates the files are of **different** sizes, This is a **problem**. It indicates that somehow either the file or the Backup of it were modified somehow outside normal activity. i.e. If the backup and real file have the same Last Modified date/Time, how can they **not** have the same file size? This should be investigated to determine the reason, or, the real file should be examined for validity and if OK, then it should be re-saved, and a new Backup created.

Requesting a Restore

To request a Restore:

- Use File Manager to browse to the **\$BACKUP** folder located in the same folder of the file that needs to be Restored.
- Enter either **RESTORE | RS** or **RESTORET | RST** next to the specific backup you wish to choose to be restored. (Details below)

Note: When you use File Manager to browse the **\$BACKUP** folder, the folder may contain both normal Backup files as well as Backup files of the associated **STATE** data. This means some backups may appear as pairs of lines, which could be confusing. To alleviate this, SPFLite will automatically add a File Mask to the FM criteria so that these extra STATE entries do not appear. If you really want to see all the folder entries (say you are using totally manual Retention Control) then simply remove the filter request from the File Mask area and press Enter.

When you Browse the \$BACKUP folder you will see the Backup files created using the following format for the file name.

testdata.txt.190403-1437.190415.txt

testdata.t	The original filename
xt	
190403-1437	The yymmdd-hhmm time-stamp of the backed up file
	NOTE: This is the last modified date/time of the <u>original file</u> . NOT the date/time the <u>Backup</u> was created.
190415	The date (yymmdd) the backup was created.

file extension	The file extension of the original file is used so that the Backup file could be opened directly by it associated application without any prior manipulation to make it acceptable.
----------------	--

Types of Restore

You may request two types of RESTORE, based on the Restore command used.

RESTORE | RS

If you choose **RESTORE | RS** the file will be restored to it's original location. If the file still exists at that location, you will be prompted for permission to replace the file.

RESTORET | RST

If you choose the **RESTORE | RST** option, then the file will be restored to it's original location, **but with** the Time-stamp shown above as part of the restored filename. This enables you to restore a backup copy without overlaying / destroying the original file. This enables you to compare the old / new files for comparison purposes. If this is done, any associated STATE files are also restored to make the restored file fully editable, including the STATE data.

Backup File Format

The Backups created by SPFLite are full, 100% identical copies of the original file. No changes or additions are made. Thus, these files are, if needed, perfectly usable by their normal application program.

If the file being backed up is an SPFLite editable file, **and** there is existing STATE data for this file, the STATE file will be backed up as well into the **\$BACKUP** folder.

If the file is selected for RESTORE, any associated STATE file will be restored as well to maintain the association between the files. For information on STATE handling see ["Saving the Edit State"](#)

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

CFG File Maintenance

The following topics are available.

[CFGMaint Tool](#)
[Export](#)
[Import](#)
[Color Export / Import](#)
[Repair](#)
[Error Log File](#)
[Modifying / Editing the Export File](#)

Introduction

The SPFLite.CFG file contains all your personal preferences, color schemes, file profiles, SET tables, Keyboard configuration etc. The settings in this file are those you have slowly

customized since your first execution of SPFLite. Loss or damage to this file could mean a considerable effort to re-establish all your preferences.

Although a Backup of the file can of course be done using normal backup tools, there are many users who miss the ability to examine and modify the settings by looking at the old INI files that used to contain all this data.

Modifying the INI files in this way was always discouraged since there was no validation performed that changes were indeed valid. But if you were careful it could be done. The same caveat applies to CFGMaint Export files. If you edit these files and run into problems, you're on your own!

CFGMaint Tool

Welcome to CFGMaint.EXE. This is a completely separate program from SPFLite itself which allows you to Export or Import CFG file data, or actually validate and repair a CFG file "in place".

Note: CFGMaint will only execute if there are **no** instances of SPFLite running, otherwise the run is aborted with an error message.

```
Syntax:   CFGMaint {  [ -EXPORT  [ table-Name ]  [ -EXPMAX
nn]  [ -QUIET]  ]
                | [ -IMPORT  [ file-name ]
                | [ -REPAIR  ]
                | [ -EXPCOLOR instance-name ]
                | [ -IMPCOLOR instance-name ]
                }

```

If no function parameters are specified -EXPORT -EXPMAX 3 -QUIET will be assumed. If -EXPORT and no -EXPMAX is provided, the last used EXPMAX value will be used. (or the default of 3)

Lets look at its various functions.

Export

The Export function will extract all data from the production SPFLite.CFG file, validate it, and then format and write the data to a normal Windows TXT file. This file can be Browsed (or Edited) like any other TXT file.

If validation detects invalid entries, their value will be corrected if the data-item is correct; and if the data-item is invalid the whole item will be dropped. All such activity will be recorded on the LOG file.

The format is similar to that of a normal INI file. Although no details are provided for each parameter, anyone familiar with the INI file format will find it quite simple to follow. Each table from the CFG file will be formatted as:

```
[TableName]
Data-item-1
Data-Item-2
.
.
=====

```

The format of the Data-Item varies depending on the type of table.

The first character of each Tablename indicates the type of CFG file table.

- **O** - An Option table for an SPFLite Instance
- **K** - A Keyboard mapping table
- **P** - A file Profile table
- **S** - A SET table
- **R** - A Retrieve table
- **B** - A Backup control table
- **E** - An Extended File table

If you have never used multiple SPFLite [Instances](#), then the names will be ODEFAULT, KDEFAULT, SDEFAULT etc.

To invoke an Export, run CFGMaint with the following command line.

```
CFGMaint -EXPORT [ table-name ] [ -EXPMAX nn ] [ -
QUIET ]
```

This will create the Export file in the normal SPFLite Home folder (the one that holds your CFG file. The file will be dated and timestamped in this format.

```
CFGMaint Exp 2020-10-27 13.42.TXT
```

or, if *table-name* was provided

```
CFGMaint Exp 2020-10-27 13.42.[table-name].TXT
```

Because the file is timestamped, SPFLite can also provide automatic file maintenance. This is controlled with the optional **-EXPMAX nn** operand.

If **-EXPMAX 0** is specified, SPFLite will simply create the Export file, it is up to you to maintain them.

If **-EXPMAX n** is specified, then SPFLite will keep **n** versions of the Export file, older versions will be automatically deleted. The default for **-EXPMAX** is 3. **Note:** Export files for specific table-names are exempt from retention processing.

Note: The last used **EXPMAX** value is saved and will be re-used until a run with a different EXPMAX value is specified.

The **-QUIET** option is used when CFGMaint is run unattended and will suppress any error Pop-Up windows. Errors will always be written to the LOG file which will be created as

```
CFGMaint Log 2020-10-27 13.42.TXT
```

In fact the default for CFGMaint, if executed without any command line operands is-
EXPORT -EXPMAX 3 -QUIET

An execution of CFGMaint to create an Export file can be easily scheduled via Windows Task Scheduler at whatever time and interval you prefer.

Import

Import is obviously the opposite of Export. It is invoked using a command line operand of **IMPORT**. If you do not provide the optional specific file-name on the command line, CFGMaint will prompt you to select the particular Export file you wish to Import.

Import will validate the contents of the file, reporting via a pop-up each error it finds and providing you with correction options. After validating the entire file, it will then prompt you again for permission to actually load the data into the CFG file.

Note: When a specific file-name is provided, CFGMaint will treat the run as a **-QUIET** run, no pop-ups for errors will be done nor will you be prompted to proceed with the actual table loading.

Import will reload data into the current production CFG file. Each table in the Import file will completely replace the table in the CFG file.

If no CFG file exists, Import will create one.

If there are no System Registry entries indicating the location of the SPFLite folders, the CFG file will be created in the normal default location.

Users\You\Documents\SPFLite

Color Export / Import

The CFG file contains all your screen color choices (amongst all the other options). There are times when you may want to copy your color 'theme' without also copying all the other options within the CFG file.

e.g.

- To copy between multiple [Instances](#) within the CFG file (if you use multiple Instances).
- To share your color theme with other SPFLite users.

CFGMaint recognizes two commands specifically created to support this need.

-EXPCOLOR	<i>instance-name</i>	to export current colors to an Export file
	and	
-IMPCOLOR	<i>instance-name</i>	to import colors from an Export file into a specific Instance

For users who do not use multiple Instances, the *instance-name* will always be **ODEFAULT**. This operand must always be specified, even though it is the normal default.

For users who **do** use multiple instances, *instance-name* should specify the exact Instance to be used for the Export / Import activity.

-EXPCOLOR will export all color related settings to a file in the normal SPFLite Home folder (the one that holds your CFG file). The file will be dated and timestamped in this format.

CFGMaint Exp 2020-10-27 13.42.[*instance-name* Colors].TXT

-IMPCOLOR will validate that the instance-name currently exists in the CFG file (it must) and then will prompt you to select the Export file which is to be imported. Note: the selected file must be a valid file created by an EXPCOLOR run, it will not process any other Export files created by the normal _EXPORT command.

Repair

CFGMaint has a **-REPAIR** option. In this mode it will examine all tables in the CFG file and perform validation of the entries. If errors are detected it will display a pop-up describing the error and asking for your permission to perform the correction. The needed changes are done to the production CFG file.

Note: If you run REPAIR and have been a long time SPFLite user, you will undoubtedly receive numerous pop-ups related to Options and Profile settings that have been obsoleted over the years. It will offer to delete these entries, in this case you should allow it to do so.

Error Log File

CFGMaint maintains a log of its activity during each run. If no errors are encountered, the log file is not even created. If errors are encountered, the log file is written to the same folder as the Export files. It is named

CFGMaint Log 2020-10-27 13.42.TXT

If created by an Export run, the timestamp will match the Export file being processed.

Modifying / Editing the Export file

Just as with INI files, we do not recommend editing the Export and then Importing it to make bulk changes to the CFG file.

Realistically though, we know many of you will still consider doing so. So here are some points to consider, in no particular order.

Be careful!

- The modified Export file need not contain all tables from the CFG file. It is fine for it to contain as little as one table (or even none!).
- You can create Export files for a single CFG table using the **EXPORT** *table-name* option.
- Normally, each Options and Profile table's entries should be complete, if you provide only a selected subset of the various entries, the line items you do not specify will be set to the SPFLite default. **Note:** For Profiles this is **not** the values from the standard Default Profile. You can **not** update single entries within a table and leave remaining entries with their current values (unless you provide these unmodified entries).

The table in the CFG file is emptied and then reloaded with the contents of the Export file and missing entries are set to their default.value.

- The delimiter line of ===== must be present at the end of each table (even if only one table is present).
- Option and Profile tables are entirely keyword=value style, like the old INI files.
- The SET, BKP and RTR tables have a count as the first line, this must match the number of entries. The entries must be numbered appropriately.
- The various tables have different characteristics which **must** be adhered to.

- Keyboard table is fixed and must not be re-arranged, the order is critical
 - The "O" and "P" tables (Options and Profiles) are old INI style, the order is not important.
 - The Retrieve table - order doesn't matter.
 - The BKP table should be left alone, there is nothing to be gained via modification but trouble.
- You may create new tables (like Profiles), and they will be loaded. If you create SET, KBD, BKP or RTR tables, it is up to you to link them to their proper Option table.

Created with the Personal Edition of HelpNDoc: [Experience the power of a responsive website for your documentation](#)

Clipboard, Cut and Paste

Contents of Article

[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 old IBM ISPF naming convention for this command, even if it is a bit "inaccurate" of a term. 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 logically 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.

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**".

Temporary Permanent Clipboards

No, that's not a typo, one problem with storing all Private Clipboards permanently is just that - they're permanent. This is fine for a lot of clipboards containing boilerplate type data, you will be using them over and over.

But many times, you just need the clipboard data to stay around for a day or two. Eventually, the \CLIP folder becomes cluttered with clipboard data that you have simply 'forgotten' about. If you prefix your clipboard name with an underscore "_", then these clip files will be cleaned up automatically after two days.

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 REPLACE 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.)

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.

You may also highlight a large span of full lines using the [T/TT](#) line command.

Once selected, the data can be Copied with the [\(Copy\)](#) function (Ctrl-C), Cut with the [\(Cut\)](#) function (Ctrl-X) or Lifted with the [\(Lift\)](#) function.

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 overwrite 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.

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(<i>name</i>) and ^S(<i>name</i>)	This variable will be replaced by the current contents of the defined SET variable called <i>name</i> . It is an error if the SET variable called <i>name</i> is undefined. ~S returns the value of the variable as-is, and ^S returns the value in upper-case.
~W and ^W	<p>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.</p> <p>e.g. X ALL; F ALL '~W'</p> <p>If this were assigned using KEYMAP to the Ctrl-F key then this could be used as follows:</p> <ul style="list-style-type: none"> ○ 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. <p>Similarly,</p> <p>FIND FIRST '~W'</p> <p>would, following the same instructions, scroll the screen to the first occurrence of the 'word' under the cursor.</p>
~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.

Command Pad

The "Command Pad" is basically a 'private' clipboard with the reserved name of CMDPAD. It's purpose is to hold a series of Keyboard commands that can be quickly used as a Keyboard Macro.

Often there is a need to record and 'play back' a series of KB commands, but there is no long term need for the commands. So the normal procedure of doing a Keyboard (Record) function, then going in to KEYMAP to assign the sequence to a specific key-code, and then later to perhaps remove the KB assignment, is just "too much bother".

CMDPAD is designed to make that process quick and simple.

How was this accomplished?

1. The KB primitive (Record) can now specify a private clipboard name as an operand. e.g. (Record/cb-name)
2. For CMDPAD, this would be (Record/CMDPAD). When recording is complete, the recorded string will be immediately available in the CMDPAD.CLIP file.
3. A new KB primitive is now available - (CmdPad) - which will inject the contents of the CMDPAD.CLIP file into the keyboard command stream.
4. Assuming the following keyboard definitions (suggested assignments only) :

Key	Assignment
SCRLK	- (Record/CMDPAD)
KPENTER	- (CmdPad)
A-KPENTER	- CLIP CMDPAD
S-KPENTER	- CUT ALL REP CMDPAD

- You can record a KB sequence by pressing SCRLK, typing the sequence, and then pressing SCRLK a 2nd time.
 - You can then **immediately** 'play back' the sequence by pressing KPENTER.
 - You can edit the current CMDPAD contents by pressing Alt-KPENTER
 - You can copy the data from any other Edit session by pressing Shft-KPENTER
5. When (Record) places a recording into CMDPAD, it does so as one long string. This is designed that way as in the past a recording was typically copied to a KEYPAD key definition, so it **had** to be a single line. When placed in a CMDPAD and edited, this is most unfriendly. SPFLite has included a macro in it's distribution which will expand the single line string into individual lines. The macro is called **CmdPadSplit**.

Crash Handling

Although nobody 'plans to crash', we have to face the fact that no program is created without bugs. Some bugs just cause incorrect functioning, but others are more serious and will "crash" the program.

In normal circumstances, Windows will typically shut down a crashed program instantly since it has no real knowledge of what the program was doing, nor how to mitigate the damage an instant shut-down will cause.

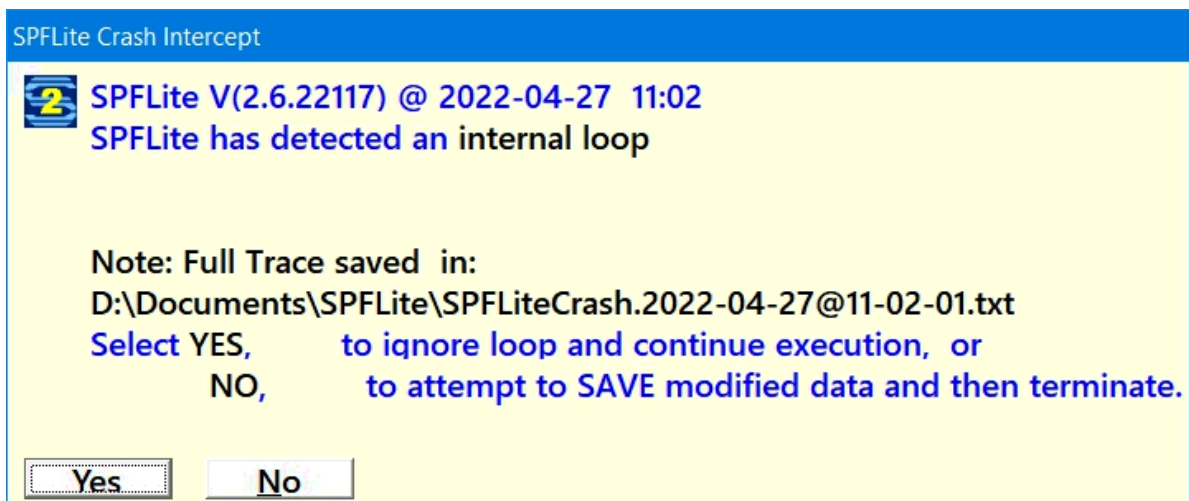
Programs can however provide routine(s) to intercept the crash and attempt to provide a graceful shut-down.

SPFLite has included this kind of support. It attempts to support two different kinds of program crashes:

Program Looping

SPFLite will monitor the various actions a user requests (typing, issuing commands, running macros etc) and watch for actions which are taking an excessive amount of time (10 seconds). You may request this activity be suspended via the command line operand **NOLOOP** or via the Primary command **LOOPCHECK**.

If monitoring is active and a loop is detected, a pop-up will appear:

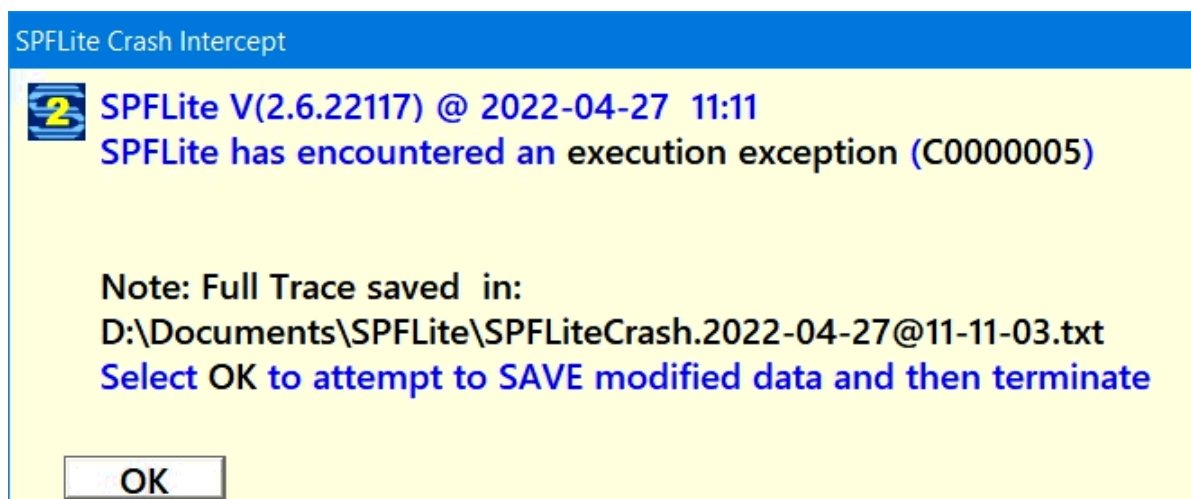


If you select **YES**, SPFLite will allow the action to continue running (although if it **is** a loop, in 10 seconds the message will re-appear).

If you select **NO**, SPFLite will attempt to **SAVE** the data from all tabs which are in Modified state before terminating.

Program Crash

If Windows informs SPFLite of an abnormal termination, this pop-up will appear:



When you select **OK**, SPFLite will attempt to SAVE the data from all tabs which are in Modified state before terminating.

Note:

Recovering from a crash is not a simple standard procedure. All recovery efforts are on a 'best efforts' basis. It is not a guaranteed successful process.

Created with the Personal Edition of HelpNDoc: [Keep Your PDFs Safe from Unauthorized Access with These Security Measures](#)

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 **(New)** 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 (New), 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.

up to and including the Suffix. Please remember that when doing RFIND, the start scan is 1 character to the right of the previous found location, **NOT** 1 character to the right of the found string.

Missing Suffix

This depends on whether the Prefix string occurs again - further right in the line.

1. If it does, the returned string will be from the start of the found Prefix to the character prior to the next occurrence of the Prefix.
2. If it does **not**, the returned string will be from the start of the found Prefix to the end of the line.

The above description is specified in terms of normal forward searches. If doing reverse searches (for example FIND LAST and then RFIND) the comments for Suffix and Prefix are reversed as the scan is working in the other direction.

WARNING: If you are going to use either of the missing Prefix / Suffix modes, **experiment first** to make sure you understand what is happening. The results are **not intuitively obvious**.

Example Searches

This example finds quoted strings'

Command > FIND D'"|"'

```
*****  ----+----1----+----2----+----3----+----4----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Result:

```
*****  ----+----1----+----2----+----3----+----4----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Note the use of single quotes to create the literal containing double quotes.

This next example is similar, but shows the use of column range operands

Command > F D'(|)' ALL 15 30

```
*****  ----+----1----+----2----+----3----+----4----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Result:

```
*****  ----+----1----+----2----+----3----+----4----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Note that the string (2) was not found because it was outside the specified column bounds of 15-30.

This next example shows the use of a missing Suffix in the D literal

```
Command > F D' ( | '
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Result:

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

Note that because the Prefix string (occurred again further right, the found string stops at the preceding character.

If an RFIND was issued at this point

Result:

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001 Say "Hello" to everyone in
000002 apartments (2) and (4)
000003 Phone: 1-212-555-1234
```

The RFIND now selects from the (to the end of line since no further occurrences of (are found.

Created with the Personal Edition of HelpNDoc: [Keep Your PDFs Safe from Unauthorized Access with These Security Measures](#)

DIFF Compare

SPFLite provides a built-in DIFF file compare tool. While this is not as comprehensive as an external DIFF tool, like KDiff3, CSDiff or WinDiff, it should prove quite useful for most file compare operations.

[DIFF Features](#)

[Invoking DIFF with an Options Dialog](#)

[Invoking DIFF via the command line](#)

[The DIFF 1 Column Output description](#)

[The DIFF 2 Column Output description](#)

Features:

- If the files being compared are already loaded into SPFLite tabs, then the entire operation is performed with no external I/O and is extremely fast.
- The Output report is displayed using normal SPFLite Edit display. This means that all normal editing commands are available to you (EXCLUDE, CUT, PASTE etc.)
- DIFF reports are created as temporary CLIP files and will thus be kept available for two days in the \CLIP folder. The temporary CLIP name will include the names of the two files being compared, date/time of the run and View type.
- Retrieval of these 'kept' reports can be done with a **DIFF LIST** command which will allow you to choose between available reports.
- User control of the color highlighting of deleted and inserted lines. [Options => Screen](#)

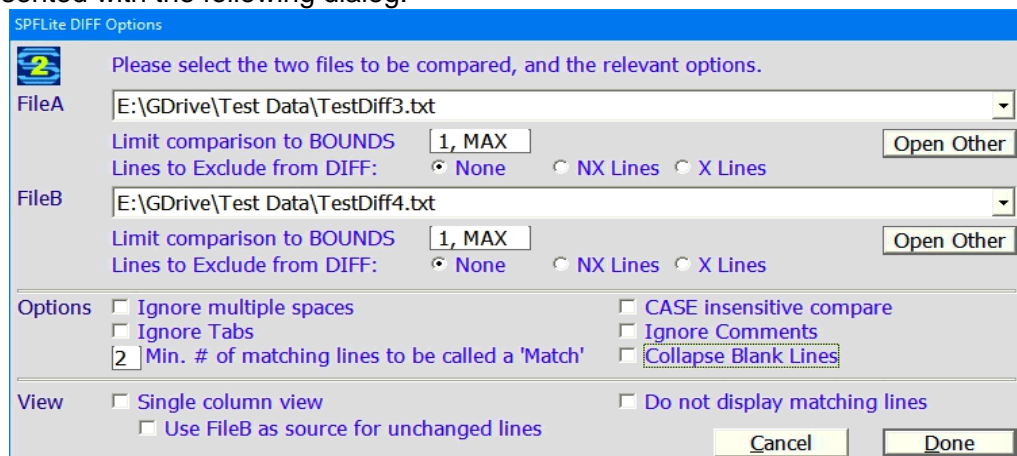
- has options to allow you to choose exactly how you would like the report to be colored.
- DIFF options to choose 1 column or 2 column format output.
- Difference lines are automatically marked as [User Lines](#), meaning a LOCATE U command can be used to quickly scroll down to the next difference. Note: LOCATE also will accept LOCATE DIFF as an alias as it seems more 'logical' to think of it that way.
- Flexible algorithm matching criteria. This can help in grouping line changes into larger 'chunks' which can visually improve the comparison.
- Ability to suppress displays of lines which match.
- Ability to exclude selected lines from the compare (e.g X or NX lines).
- Ability to compare only within specified BOUNDS for each file.
- Ability to compress multiple spaces down to 1 space before the compare.
- Ability to 'collapse' multiple consecutive blank lines down to a single line.
- Ability to do a case-insensitive compare.
- Ability to ignore embedded TAB characters if present.
- Ability to ignore Source Code comments if the files have valid AUTO colorization files

Invoking a DIFF Compare

SPFLite allows you to invoke DIFF and specify the files and options to be used in a variety of ways. You may choose whichever suits your own working style.

Options Dialog

This option if invoked with a simple **DIFF** command, with no operands. You will be presented with the following dialog:



FileA and FileB

These input boxes will be populated with a list of the currently Open files in SPFLite. You may select the files you wish to compare from the drop-down list.

If a desired file is not currently Open, click the **Open Other** button, a normal file selection dialog will appear and you can locate and select your file. The file will be loaded into a new Edit Tab.

Limit comparison to BOUNDS

You may specify the Bounds within which the comparison will be made, Enter the Left and Right Bounds as two values separated by a comma. See [BOUNDS](#) for details on Bounds usage.

Lines to Exclude from Diff:

Specify what lines (if any) you wish to *exclude* from the DIFF comparison. You may select **None**, **NX Lines** or **X Lines**.

Options

The following options for the DIFF comparison may be chosen, they are all independent and may be chosen in any combination.

Ignore multiple spaces

This will treat all strings of multiple spaces as if they are a single space.

Case insensitive compare

This will do all compares ignoring differences in text case.

Ignore Tabs

This will treat all embedded TAB characters as if they are a single space.

Ignore Comments

This will, if the file type has as associates AUTO file, cause the comments in Source text to be ignored.

Comment types where the remainder of a line is a comment, will be truncated.

Bracketed comments (like /* ... */) will be replaced with spaces.

Min. # of matching lines to be called a 'Match'

This number is used when the DIFF process is attempting to re-synchronize lines following some deletions or insertions. It indicates how many **consecutive** matching lines must be found before considering things are 'back in sync'. The default is 5, which for most program source type text files seems to produce the best report. This is a personal preference item which you should experiment with to find what works best for **you**.

Collapse Blank lines

If selected, a consecutive sequence of blank lines in a file will be reduced to a single blank line. If you are also using "Lines to Exclude from Diff" (above), then the collapse of blank lines is done **after** the X/NX filtering is done.

View

These options allow you to select the formatting of the DIFF output report.

Single column view

Select this to request a Single Column report, un-select to request a Two-Column report. See the descriptions of the two report types below.

Use FileB as source for unchanged lines

When 1Column view is selected, and the FileA and FileB lines are considered a match, the report only shows the data from one file. After all, they're supposed to match.

But when options like Case Insensitive, Space compression, Comment removal etc. are used, the FileA and FileB data may actually be somewhat different.

Normally, DIFF will use the FileA data to display, this option allows you to request that the display show the FileB data instead.

Do not display matching lines

If Selected, the DIFF report will not display lines considered to be a match, only differences.

Press **Done** when you have made your choices.

Other than the File Names, all the option settings will be retained and used as the defaults for future DIFF requests.

An exception to this is the FileA and FileB Bounds values, these will be saved only

during the current day and will be reset to 1,MAX the next day.

Using the DIFF command with operands

The format of the DIFF command in full is:

```
DIFF      { [ FileA-TabNum | FileA-filename ] }
          { [ FileB-TabNum | FileB-filename ] }
```

OR

```
DIFF LIST
```

Once you enter the command, you will still be presented with the Option dialog described above where you can choose the other possible options for the DIFF operation.

Operands

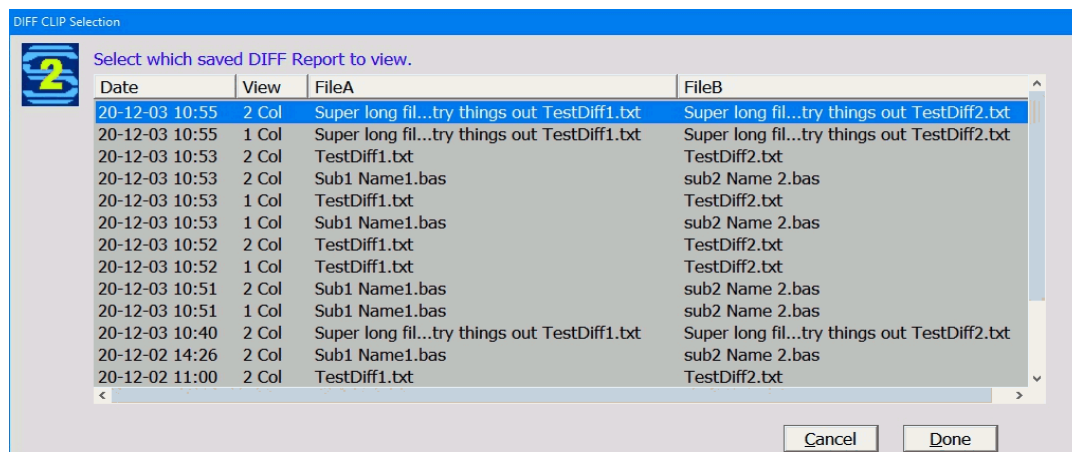
```
[ FileA-TabNum | FileA-filename ]
[ FileB-TabNum | FileB-filename ]
```

DIFF compare requires two files, these operands allow you to specify the file in one of three ways, whichever is more convenient for you.

Filex-TabNum	You may enter a .simple number to indicate the Tab # currently containing the desired file. The File Manager tab is 1, the next tab to the right is 2 etc. Tab 1 (FM) is not allowed.
Filex-FileName	You may enter the full name of the desired file. This can be a file currently open in one of the tabs, or some other file not currently Open. If a file is not currently Open, it will be loaded into a new tab prior to the DIFF compare.

LIST

If LIST is the only operand, you will be presented with a pull-down list of all available DIFF reports you have created in the last couple days. Simply choose the one desired and it will be opened for review.



Invoking DIFF from the File Manager

You may invoke DIFF in File Manager by selecting the two files to be compared using the **DFA** and **DFB** line commands.

The DIFF 1 Column Output description

The 1 column output report shows the line numbers of the data in the FileA and FileB files on the left hand side. Where a line exists on one side and not on the other, the line number is replaced by ----- or ++++++. The text for all lines is complete and if needed, the screen can be scrolled to the right to view all of it. Here's a sample report showing the various types of differences.

Note lines 9-10, 15-16, 22 and 27-29 are all marked as [User Lines](#). For large difference reports this means you can use a LOCATE U (ot LOCATE DIFF) command to quickly scroll down to the next actual difference.

```

CLIP - SPFLite(v2.2.20335)
File Manager | TestDiff1.txt | TestDiff2.txt | TESTDIFF1.TXT~TESTDIFF2.TXT
Command > Scroll > CSR

====COLS> ....._-----1-----2-----3-----4-----5-----6-----7--
***** Top of Data *****
00000001 ***** SPFLite DIFF Report *****
00000002 FileA: - 2020-11-30 11:15:46 - E:\GDrive\Test Data\TestDiff1.txt
00000003 FileB: - (Newer) 2020-11-30 11:32:55 - E:\GDrive\Test Data\TestDiff2.txt
00000004 Options - Match criteria = 2 lines
00000005 FileA FileB ***** Text Data *****
00000006 Number of mis-matches: 4
00000007
00000008 00001 00001 File Record 1
00000009 00002 ----- File Record 2
00000010 +++++ 00002 File Record 2 Modified
00000011 00003 00003 File Record 3
00000012 00004 00004 File Record 4
00000013 00005 00005 File Record 5
00000014 00006 00006 File Record 6
00000015 +++++ 00007 File Record 7
00000016 +++++ 00008 File Record 8
00000017 00007 00009 File Record 9
00000018 00008 00010 File Record 10
00000019 00009 00011 File Record 11
00000020 00010 00012 File Record 12
00000021 00011 00013 File Record 13
00000022 00012 ----- File Record 13 Deletion
00000023 00013 00014 File Record 14
00000024 00014 00015 File Record 15
00000025 00015 00016 File Record 16
00000026 00016 00017 File Record 17
00000027 +++++ 00018 File Record 17a Added
00000028 +++++ 00019 File Record 17b Added
00000029 +++++ 00020 File Record 17c Added
00000030 00017 00021 File Record 18
00000031 00018 00022 File Record 19

Clip | L 00000001 C 1 | Lines: 31 | Cols 1 to 84 | Bnds: MAX | INS | DS | Line Len 0031 | ANSI

```

Lets look at some lines in detail.

```

00000020 00010 00012 File Record 12
00000021 00011 00013 File Record 13
00000022 00012 ----- File Record 13 Deletion
00000023 00013 00014 File Record 14
00000024 00014 00015 File Record 15
00000025 00015 00016 File Record 16

```

Here we can see that FileA Line 00012 is not present in FileB (the ----- line number). The line is highlighted as a deletion according to what was specified in [Options -> Screen](#).

The lines above and below the deleted lines are present in both FileA and FileB as there are valid line numbers for each line in each file. These lines are displayed in your normal text colors.

Here's another detail:

```

00000026 00016 00017 File Record 17
00000027|+++++ 00018 File Record 17a Added
00000028|+++++ 00019 File Record 17b Added
00000029|+++++ 00020 File Record 17c Added
00000030 00017 00021 File Record 18
00000031 00018 00022 File Record 19

```

We can see lines 00018, 00019 and 00020 in FileB have no matching line in FileA - they are all +++++. This is an insertion and is colored as per the setting in Options => Screen. Again the lines before and after the insertion lines are normal, unchanged matching lines.

One more detail:

```

00000005 FileA FileB *****
00000006 Number of mis-matches: 4
00000007
00000008 00001 00001 File Record 1
00000009|00002 ----- File Record 2
00000010|+++++ 00002 File Record 2 Modified
00000011 00003 00003 File Record 3

```

This shows a simple modification of a line. In order to show you both versions of the modified line, it takes **two** lines in the report. They are colored as a Deletion followed by an insertion. Note that both FileA and FileB line numbers are the same - 00002

The DIFF 2 Column Output description

The 2 column output report shows the lines from the two files side by side, which is normally the simplest visual organization to understand. Where a line exists on one side and not on the other, the line number is blanked. SPFLite will adjust the widths of the two columns to the maximum that can fit in your displayed screen width. Because these are fixed-width, non-scrollable columns, only as much data as will fit is shown. Side scrolling of the columns is not supported, so it is advisable to make your screen as wide as possible (Full Screen?)

Here's a sample report showing the various types of differences. This is the exact same file compare that was shown above in the 1 column description.

Note lines 9, 14-15, 21, and 26-28 are all marked as [User Lines](#). For large difference reports this means you can use a LOCATE U command to quickly scroll down to the next actual difference.

```

CLIP - SPFLite(v2.2.20335)
File Manager | TestDiff1.txt | TestDiff2.txt | TESTDIFF1.TXT~TESTDIFF2.TXT
Command > Scroll > CSR

=====COLS> .....-+---+---1---+---2---+---3---.....-+---1---+---2---+---3---
***** ***** Top of Data *****
00000001 ***** SPFLite DIFF Report *****
00000002 FileA: - 2020-11-30 11:15:46 - E:\GDrive\Test Data\TestDiff1.txt
00000003 FileB: - (Newer) 2020-11-30 11:32:55 - E:\GDrive\Test Data\TestDiff2.txt
00000004 Options - Match criteria = 2 lines
00000005 ***** FileA Data ***** ***** FileB Data *****
00000006 Number of mis-matches: 4
00000007
00000008 00001 File Record 1 | 00001 File Record 1
00000009 00002 File Record 2 ~ 00002 File Record 2 Modified
00000010 00003 File Record 3 | 00003 File Record 3
00000011 00004 File Record 4 | 00004 File Record 4
00000012 00005 File Record 5 | 00005 File Record 5
00000013 00006 File Record 6 | 00006 File Record 6
00000014 | > 00007 File Record 7
00000015 | > 00008 File Record 8
00000016 00007 File Record 9 | 00009 File Record 9
00000017 00008 File Record 10 | 00010 File Record 10
00000018 00009 File Record 11 | 00011 File Record 11
00000019 00010 File Record 12 | 00012 File Record 12
00000020 00011 File Record 13 | 00013 File Record 13
00000021 00012 File Record 13 Deletion <
00000022 00013 File Record 14 | 00014 File Record 14
00000023 00014 File Record 15 | 00015 File Record 15
00000024 00015 File Record 16 | 00016 File Record 16
00000025 00016 File Record 17 | 00017 File Record 17
00000026 | > 00018 File Record 17a Added
00000027 | > 00019 File Record 17b Added
00000028 | > 00020 File Record 17c Added
00000029 00017 File Record 18 | 00021 File Record 18
00000030 00018 File Record 19 | 00022 File Record 19
***** ***** Bottom of Data *****
Clip | L 00000001 C 1 | Lines: 30 | Cols 1 to 84 | Bnds: MAX | INS | DS | Line Len 0031 | ANSI

```

Lets look at some lines in detail.

```

00000019 00010 File Record 12 | 00012 File Record 12
00000020 00011 File Record 13 | 00013 File Record 13
00000021 00012 File Record 13 Deletion <
00000022 00013 File Record 14 | 00014 File Record 14
00000023 00014 File Record 15 | 00015 File Record 15

```

Here we can see that FileA Line 00012 is not present in FileB (the right side is blank). The center divider is a < character used to highlight FileA deletions. The line is also highlighted as a deletion according to what was specified in [Options -> Screen](#).

The lines above and below the deleted lines are present in both FileA and FileB as there are valid line numbers for each line in each file. These lines are displayed in your normal text colors and the center divider is a | character.

Here's another detail:

```

00000025 00016 File Record 17 | 00017 File Record 17
00000026 | > 00018 File Record 17a Added
00000027 | > 00019 File Record 17b Added
00000028 | > 00020 File Record 17c Added
00000029 00017 File Record 18 | 00021 File Record 18

```

We can see lines 00018, 00019 and 00020 in FileB have no matching line in FileA - the left side is blank. This is an insertion and is colored as per the setting in Options => Screen. The center divider is a > to indicate an insertion. Again the lines before and after the insertion lines are normal, unchanged matching lines.

One more detail:

```

00000005 ***** FileA Data *****
00000006 Number of mis-matches: 4
00000007
00000008 00001 File Record 1 | 00001 File Record 1
00000009 00002 File Record 2 ~ 00002 File Record 2 Modified
00000010 00003 File Record 3 | 00003 File Record 3
00000011 00004 File Record 4 | 00004 File Record 4
00000012 00005 File Record 5 | 00005 File Record 5

```

This shows a simple modification of a line. They are colored as a Deletion / Insertion and the center character is a ~

Two Column Center Divider Characters

The use of different characters in the center divider column is to provide easy selection of the different line types. Since the DIFF report is displayed using a normal CLIP window, you have the full range of SPFLite commands available to Exclude Lines, Delete any unwanted lines, etc.

Created with the Personal Edition of HelpNDoc: [Streamline Your Documentation Creation with a Help Authoring Tool](#)

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.

Enumerations

Contents of Article

[Combining Fast Enumeration with Line Exclusion](#)
[Enumeration patterns](#)
[Examples Of ENUM](#)
[Handling of blanks in the prefix](#)
[Use of the ENUMWITH command](#)

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.

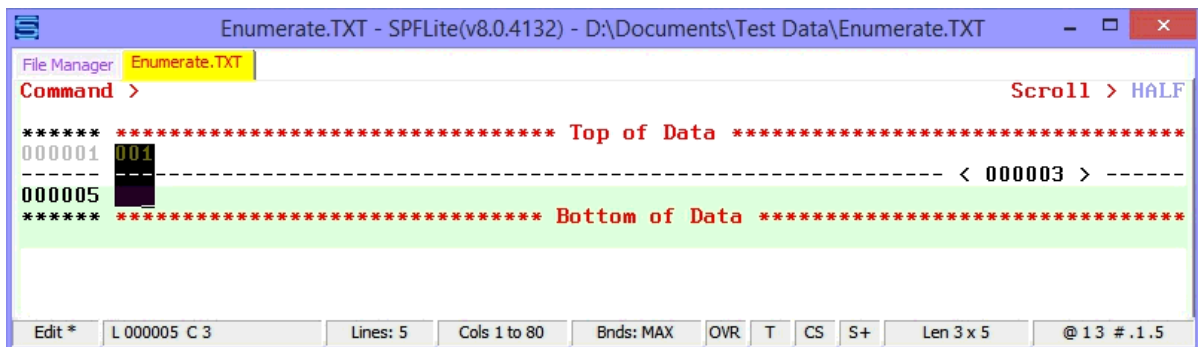
The highlighted field on the first line of selected text constitute the '*initial*' or '*base*' value of the enumeration, defining the first value in the sequence. All subsequent lines (in increasing line number order) in the line range are incremented for each new line.

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**.

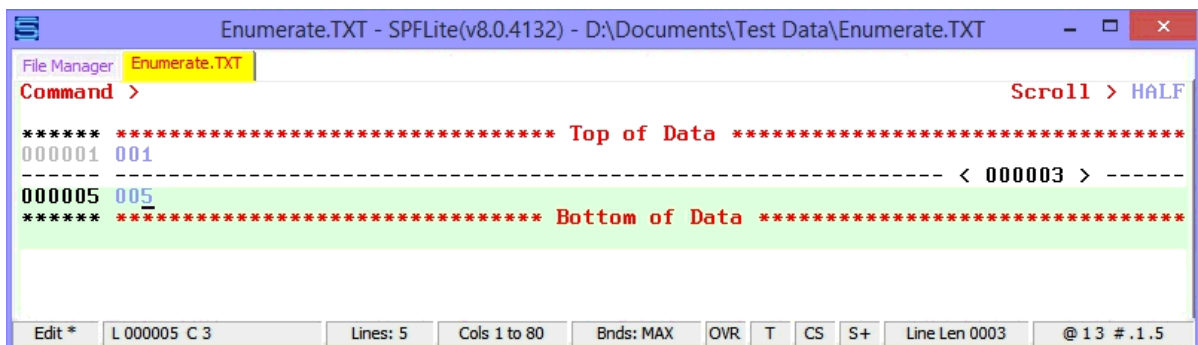
Combining Fast Enumeration with Line Exclusion

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.



Then, issue the desired enumeration function as you have mapped it. You will see the final line containing 005 as expected:



Finally, unexclude the lines to see all of them:

```

Enumerate.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Enumerate.TXT
File Manager Enumerate.TXT
Command >
Scroll > HALF

***** Top of Data *****
000001 001
000002 002
000003 003
000004 004
000005 005
***** Bottom of Data *****

Edit * Lines: 5 Cols 1 to 80 Bnds: MAX OVR T CS S+

```

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 of ENUM

- 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*):

<code>...&Hff</code>	enumerates as	<code>...&Hff</code>	<code>..&H100</code>	<code>..&H101</code>	etc.
<code>Page..8</code>	enumerates as	<code>Page..8</code>	<code>Page..9</code>	<code>Page.10</code>	etc.
<code>..(98)</code>	enumerates as	<code>..(98)</code>	<code>..(99)</code>	<code>.(100)</code>	etc.
<code>(..98)</code>	enumerates as	<code>(..98)</code>	<code>(..99)</code>	<code>(.100)</code>	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.

Excluded Lines

Contents of Article

[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.

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:

```

EDIT          MYLIB.TEST.DATA(ONE)  - 01.30
Columns 00001 00072
Command ==>
Scroll ==> HALF
***** Top of Data
*****
000001 LINE ONE

- - - - - 2 Line(s)
not Displayed
000004 LINE FOUR
***** Bottom of Data
*****

```

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.

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.

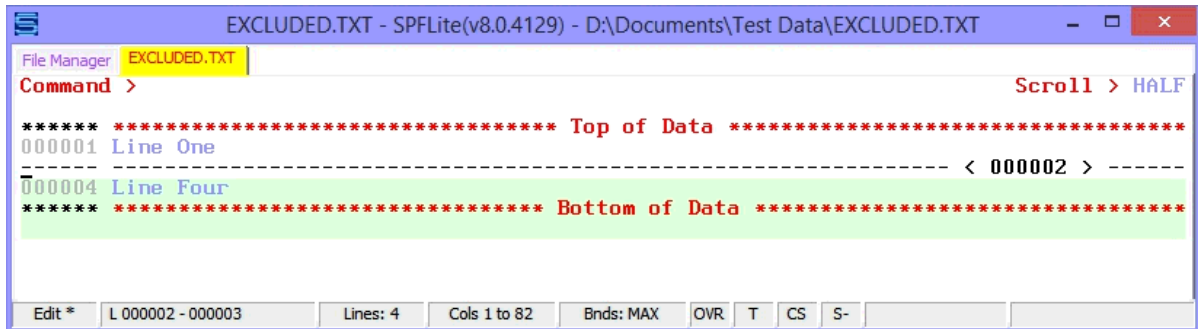
Wait a minute! Didn't we just say *not* to use **DD**? Well, yes and no. You wouldn't use **DD** to delete *just* the lines represented by the placeholder. But, you *can* use a **DD** block where excluded-lines placeholders are one or both of the *ends* of the **DD** block. When you do this, all the 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

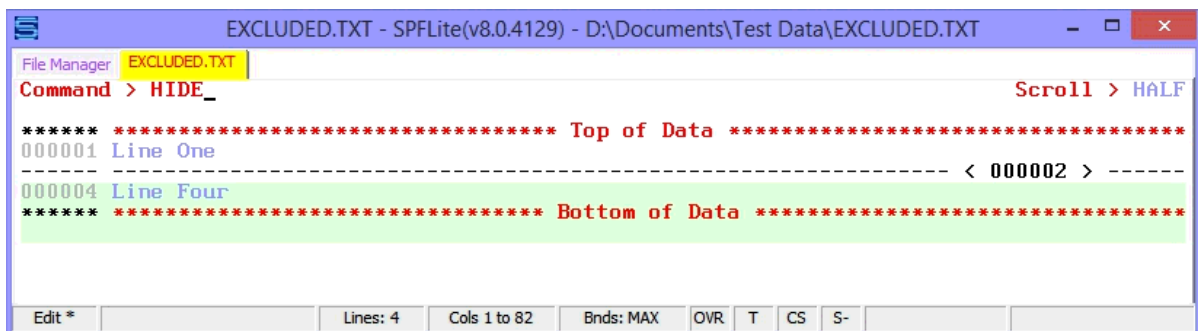
The ISPF version of the placeholder display is fine for new users unfamiliar with the concept of using excluded lines, and it looks good in a help document. Prior versions of SPFLite also used a similar display. But, once you get the hang of it, IBM's format is a little wordy and distracting, especially 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:



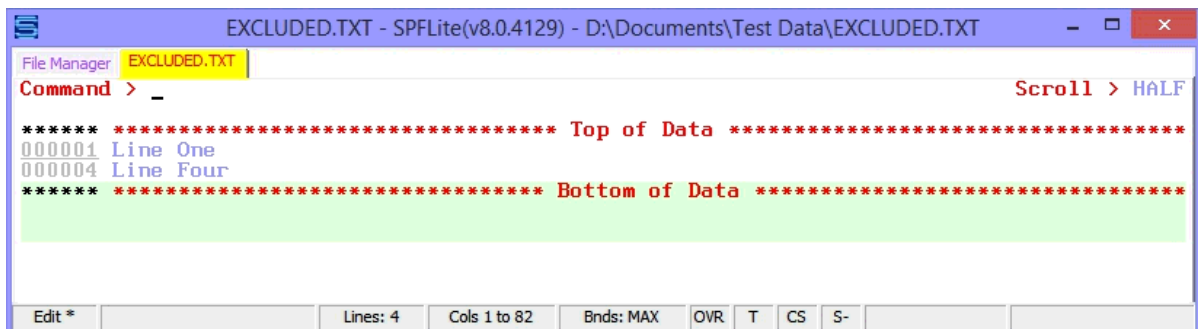
In some cases, even this brief format is too much. For that, you can use the **HIDE** command to collapse these excluded regions down to just an underline. IBM ISPF users familiar with this may be used to saying **HIDE X**. Since there is nothing else you can hide other than excluded lines, SPFLite does 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**):



After **HIDE**:



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** line command will show (unexclude) a range of lines. If an **S** command is placed on an excluded-line placeholder, the entire region is unexcluded. **S** is 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.
- The [LINE](#) primary command can be used to apply the **S** 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:

EXCLUDE	<i>only non-excluded lines considered</i>
FLIP	<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 color, 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.

```

EDIT - EXCLUDED.TXT - SPFLite(v10.0.8125) - e:\GDrive\Test Data\EXCLUDED.TXT
File Manager EXCLUDED.TXT
Command > Scroll > CSR

===COLS> ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8-
***** Top of Data *****
00000001 Line One
x2 Line Two
00000003 Line Three
00000004 Line Four
***** 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:

```

EDIT - EXCLUDED.TXT - SPFLite(v10.0.8125) - e:\GDrive\Test Data\EXCLUDED.TXT
File Manager EXCLUDED.TXT
Command > F line X DX Scroll > CSR

===COLS> ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8-
***** Top of Data *****
00000001 Line One
----- < 000002 > -----
00000004 Line Four
***** 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?

```

EDIT - EXCLUDED.TXT - SPFLite(v10.0.8125) - e:\GDrive\Test Data\EXCLUDED.TXT
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 Four
***** 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:

```

EDIT - EXCLUDED.TXT - SPFLite(v10.0.8125) - e:\GDrive\Test Data\EXCLUDED.TXT
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 Four
***** Bottom of Data *****

Edit L 00000003 C 3 Lines: 4 Cols 1 to 81 Bnds: MAX OVR T DS S-

```

And the Line/Column display on the status will now show **L 000003 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:

EXCLUDED2.txt - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED2.txt

File Manager: EXCLUDED2.txt

Command: > Scroll > HALF

```

***** ***** Top of Data *****
CC   Line One
----- < 000002 > -----
CC   Line Four
000005 // Five
A_   -- Six
***** ***** Bottom of Data *****

```

Edit * L 000006 Lines: 6 Cols 1 to 82 Brds: MAX OVR T CS S- Line Len 0006

When you are done, the file display will look like this:

EXCLUDED2.txt - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED2.txt

File Manager: EXCLUDED2.txt

Command: > Scroll > HALF

```

***** ***** Top of Data *****
000001 Line One
----- < 000002 > -----
000004 Line Four
000005 // Five
000006 -- Six
000007 Line One
----- < 000002 > -----
000010 Line Four
***** ***** Bottom of Data *****

```

Edit * L 000007 C 1 Lines: 10 Cols 1 to 82 Brds: MAX OVR T CS S- X'4C' = 76 @ 11 #.7.7

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 **S** 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:

EXCLUDED2.txt - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED2.txt

File Manager EXCLUDED2.txt

Command > Scroll > HALF

```

***** ***** Top of Data *****
CC   Line One
----- < 000002 > -----
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. 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.

EXCLUDED2.txt - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED2.txt

File Manager EXCLUDED2.txt

Command > Scroll > HALF

```

***** ***** Top of Data *****
000001 Line One
----- < 000002 > -----
000004 Line Four
000005 // Five
000006 -- Six
----- < 000004 > -----
***** ***** Bottom of Data *****

```

Edit * L 000007 - 000010 Lines: 10 Cols 1 to 82 Bnds: MAX OVR T CS S-

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-**.

EXCLUDED2.txt - SPFLite(v8.0.4129) - D:\Documents\Test Data\EXCLUDED2.txt

File Manager EXCLUDED2.txt

Command > Scroll > HALF

```

***** ***** Top of Data *****
CC-   Line One
----- < 000002 > -----
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:

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 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.

Extended File Types

Extended File Types

Contents of Article

[Introduction](#)
[Entering EFT Rules](#)
[Introduction to EFT Patterns](#)
[Selecting a Profile using a Simple Pattern](#)
[Using Profile Pattern Codes](#)

Introduction

Extended File Types help you better manage your Edit Profiles, giving you greater flexibility in determining the kind of file you are editing and which SPFLite Edit Profile gets used. EFT does this by allowing you to specify a **Pattern** that matches one or more file names, and then associating that pattern with a **Profile**.

Entering EFT Rules

To edit your current list of EFT rules, simply enter the **EFT** command on any command line. The list will be displayed as a normal edit tab labelled (EFT Edit). Alter the entries as needed.

The first time you do that, you will see a screen like this, with a comment on line 1:

```

EFT-Edit - SPFLite(v2.7.22328)
File Manager (EFT Edit)
Command > | scroll > CSR

=COLS> -----1-----2-----3-----4-----5-----6-----
***** Top of Data *****
000001 ; Enter your EFT entries following this line
***** Bottom of Data *****

EFT Edit * | Lines: 1 | Cols 1 to 64 | Bnds: MAX

```

You can enter your own full-line comments, or put a comment-mark of ; after your EFT rules. Comments and blank lines are optional and have no effect.

- This initial comment is just a "welcome message" for informational purposes
- You are free to delete it,

EFT Syntax Formats

There are two basic EFT syntax formats:

Profile Mode

***pattern* = *profilename* [,*profile-override*] ... [; *comments*]**

<i>pattern</i>	<p>To start with a simple example, the left-hand "pattern" is just an ordinary file-name extension (INC), and the right-hand "profile" is the name of the SPFLite Profile (BAS) that you want to use.</p> <pre>.INC = BAS ; PowerBasic Include files will use BASIC profile</pre> <p>The spaces around the = sign are optional; you could write the rule above like this:</p> <pre>.INC=BAS ; PowerBasic Include files will use BASIC profile</pre> <p>The <i>pattern</i> (.INC in the above example) is the simplest form of mask. EFT supports a much more powerful masking structure (See Using Profile Pattern Codes below)</p> <p>If you are are familiar with RegEx expressions, you can code your own RegEx mask and enter it for the <i>pattern</i> using the normal SPFLite format of an R"xxxxxx" literal. If you use this method, the RegEx expression must be complete and valid, SPFLite will not modify the expression in any way.</p> <p>The complete list of all known EFT Rules is called the EFT Definition List. That is what you are editing when are in an EFT Edit session. When you are done editing your EFT rules, close the EFT Edit session by using the END primary command (or just F3 if you have standard PF key mapping). The EFT rules are saved automatically when you leave the edit session.</p>
<i>profilename</i>	The name of the profile to be used for files matching this profile.
<i>profile-override</i>	<p>A profile override is simply a temporary override of one or more of the normal values that make up the <i>profilename</i> specified. Overrides are specified exactly as the normal Primary command, separated by commas when more than one is entered.</p> <p>e.g. this example uses the DEFAULT Profile, but overrides two options</p> <pre>.XPQ = DEFAULT,EOL LF,XTABS 8 ;Override CRLF to LF, XTABS to 8</pre> <p>Using overrides can significantly reduce the number of unique Profiles you may require.</p> <p>Note: When any overrides are in effect, the Profile will be treated as LOCKED so that these temporay overrides do not get saved back into the Profile.</p>
<i>;comments</i>	All characters following a ; on a line are treated as comments. A

	leading ; on a line creates a full line comment.
--	--

NonText Mode

***pattern* = NONTEXT [,OPENWITH { WINDOWS | full-file-path }] [;
comments]**

<i>pattern</i>	Same as <i>pattern</i> above.
<i>NONTEXT</i>	<p>A constant. It indicates that files selected with this pattern do not contain normal editable text.</p> <p>If no optional parameters (below) then:</p> <ul style="list-style-type: none"> a) If simply selected in File Manager, no action will be taken. b) If detected in a FF (Find in File) search, the file will be skipped.
<i>OPENWITH</i>	<p>A constant. If simply selected in File Manager, it requests that the file be opened externally from SPFLite. How it is opened is controlled by the parameter following the OPENWITH operand.</p> <p>If OPENWITH WINDOWS is specified: The file will be opened by Windows, using whatever default application has been established on the system. e.g. .DOC = NONTEXT,OPENWITH WINDOWS would open any .DOC file using the default (normally WORD)</p> <p>If OPENWITH full-file-spec is specified: The file will be opened by the specified program. This would be used, for example, when there is no default program setup, or an alternate program is desired. If the full-file-spec contains spaces, it should be enclosed in double quotes.</p>
<i>;comments</i>	All characters following a ; on a line are treated as comments. A leading ; on a line creates a full line comment.

Profile Display when EFT overrides are in effect

When a Profile is in use, and one or more values have been overridden via the EFT override method, the normal PROF display is altered to remind you of these overrides. A separate ==EFT> line is displayed showing the EFT statement in effect, and the individual values that have been overridden will be enclosed in <...> brackets to hilite them. Here's an example:

```

EDIT - SPFLite(v3.0.24020) - D:\DOCUMENTS\SPFLITE2\SOURCE\LCmd.inc
File Manager: LCmd.inc
Command > |
Scroll > CSR

==COLS> -----1-----2-----3-----4-----5-----6-----7-----8-----9-----1
***** ***** Top of Data *****
==EFT> .inc = BAS
==PROF> PROFILE BAS UNLOCKED, ACTION OFF, AUTOBKUP OFF, AUTOCAPS ON, AUTONAME BAS
==PROF> AUTOSAVE OFF PROMPT, BOM OFF, CAPS OFF, CASE T, CHANGE DS, COLLATE ANSI
==PROF> COLS ON, COMMENTS 2 67 3 100, EMACRO NONE, EOL CRLF, FOLD OFF, HEX OFF, HILITE FIND AUTO
==PROF> IMACRO NONE, LRECL 0, MACLIB NONE, MARK ON, MINLEN 0, MODE EDIT, NUMTYPE NONE
==PROF> PAGE OFF, PRESERVE OFF, RECFM U, SCROLL CSR, SOURCE ANSI, START FIRST
==PROF> STATE ON, SUBARG OFF, SUBCMD OFF, TABS ON, TABBND OFF, XFORM NONE, XTABS 0
==WORD> A-Z a-z 0-9 _
==MARK>
==MASK>
==TABS> * * * * *
==COLS> -----1-----2-----3-----4-----5-----6-----7-----8-----9-----1
==BND> <+-----1-----2-----3-----4-----5-----6-----7-----8-----9-----1
0000001 '-----
0000002 '----- License Stuff
0000003 '-----
0000004 '-----

Edit 2024-01-19 16:06 Lines: 3736 Cols 1 to 100 Bnds: MAX INS T DS S+ Profile: BAS CAPS OFF ANSI CRLF

```

Order of EFT Rules

When editing your EFT rules, you should keep in mind the order of the entries. When matching a filename against the EFT table the entries are tested **in the order** of their position in the list and the 1st entry which matches the filename is the one used.

Depending on how you code the various masks it is possible to create multiple masks that match a particular filename. Basically, the more specific the mask pattern is, the closer to the top of the list it should be placed.

Introduction to EFT Patterns

The real power of EFT comes from the use of **Patterns** to select a Profile using something other than (just) the file extension.

Let's begin with a simple, real-world example. I have text files containing song lyrics for my karaoke hobby. To make these files more convenient to use on the go, I download them to my cell phone, and use the phone's text-file reader app to see them. Only, using those files on a cell phone comes with some issues:

- The text-reader is limited to **47** columns of cell-phone screen size – an unusual, non-obvious line-length.
- The text-reader requires the use of file extension **TXT**, but assumes that all text files are **UTF8** rather than ANSI.
- SPFLite assumes TXT files are **ANSI**, and values > X'80' are **not** UTF8. If files with such characters are downloaded to my cell phone, they display incorrectly.
- I don't want to change the PC file extension from TXT to something else.

There is more than one way of solving this problem. Let's start simple.

Selecting a Profile using a Simple Pattern

I can create a new Profile named **LYRICS** to allow me to edit these files more easily. What features does this Profile need to have, to solve the problem? Here are the important settings (for me):

```

PROFILE LYRICS UNLOCKED, AUTOCAPS OFF, BOM OFF, CAPS OFF, CASE
T, COLLATE UTF8
FORMAT U CRLF 0, MARK ON, SOURCE UTF8, STATE ON, XTABS 3

```

When I first start editing these lyrics files, I will use a **MARK** command and put a > sign in column 47 of the **=MARK>** line, as a reminder not to type any further right of column 47.

To make this first example simple, I will change the name of this lyrics file from

D:\LYRICS\Billy Joel - Piano Man.txt

to

D:\LYRICS\Billy Joel - Piano Man.cel.txt

All that is needed now is to create an EFT rule that matches files ending in **.cel.txt** like this:

".cel.txt" = LYRICS

The left side of this rule contains the Pattern that will match the kinds of files that use the **LYRICS** Profile. Here are some points to remember:

- When a Pattern has punctuation or embedded spaces, you should enclose it in quotes.
- This kind of pattern is called a **Partial Pattern**, because we are not trying to reference the **entire** file name – with the owing folder(s) and the drive letter – just **part** of the name.
- When you use a Partial Pattern, it matches against the right-hand side of your file name; so it is **right-justified**. So, EFT will try to match **".cel.txt"** to the **right-hand side** of the name, not to the **entire** name.

Selecting a Profile using a Wild-Card Pattern

The example above works, but it's not an ideal solution – it forced me to change the end of the name from **.txt** to **.cel.txt** and I said I didn't want to do that. How can I avoid changing the basic name of the file, and still use EFT?

Notice the full name of this file:

D:\LYRICS\Billy Joel - Piano Man.txt

If I adopted a convention of only putting these "lyrics" files in the LYRICS directory, I could make an EFT rule for them. Suppose you were in a Windows command line, and wanted to get a list of all these files. What would you use for a **DIR** command? Something like this:

D:\>DIR \LYRICS*.TXT

Volume in drive D is SAMSUNG

Volume Serial Number is AA43-D6A8

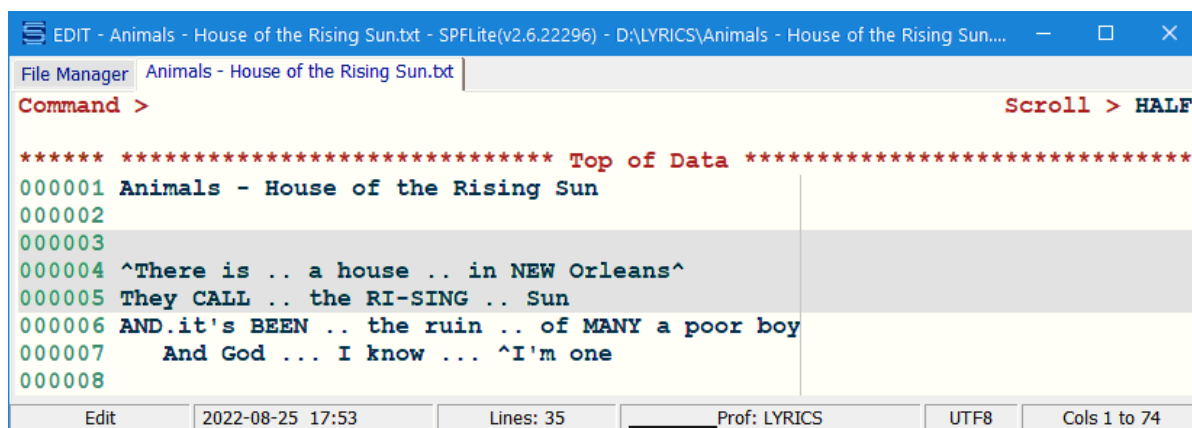
Directory of D:\LYRICS

2022-08-25	17:53	1,031 Animals - House of the Rising
		Sun.txt
2022-08-08	15:54	716 Beatles - Blackbird.txt
2022-08-13	14:54	1,464 Bee Gees - Nights on Broadway.txt
2022-08-13	14:55	1,310 Billy Idol - White Wedding.txt
2022-10-26	12:56	2,122 Billy Joel - Piano Man.txt
2022-08-13	14:55	791 Bob Dylan - Knockin' on Heaven's
		Door.txt
6 File(s)		7,434 bytes

We can do the same kind of thing with EFT. Let's make a rule to connect the **TXT** files in **this** directory to the **LYRICS** Profile, but not in any **other** directory. Using the **EFT** Primary Command, enter the EFT Edit session, and type an EFT rule line like this:

"\LYRICS*.TXT" = LYRICS

Now, when I am in the SPFLite FM screen showing the **D:\LYRICS** directory, I can click on the first **TXT** file in this folder, and sure enough – it's using the **LYRICS** profile, with a **SOURCE** type of **UTF8**:



Notice how the text on **line 6** runs right up to the **MARK** line in column **47**. That would mean on my cel phone, the text display app should be showing that line right next to the edge of the screen – and that's **exactly** how it looks.

I want a **MARK** line for these **lyrics** files, to make sure I don't create lines that can't fit on my cel phone screen, but I **don't** want a **MARK** line for editing ordinary PC **TXT** files elsewhere – since for them, column 47 is meaningless.

Using Profile Pattern Codes

In the example above, the EFT rule **"\LYRICS*.TXT" = LYRICS** has a pattern of **"\LYRICS*.TXT"**. This resembles the kind of "file name mask" that you would use in a Windows command-line screen, or within Windows File Manager. EFT allows two "wild-cards" for making patterns. In EFT terminology, a "wild-card" is called a **Pattern Code**.

- A Pattern Code of **?** means to match any one character in the file name, not including **:** or ****
- A Pattern Code of ***** means to match zero or more characters in the file name, not including **:** or ****
- A Pattern Code of ****** means to match zero or more characters in the file name. The ****** code is something not available in Windows file masks. ****** matches everything ***** does and **also** matches **\ path separators**. Even though ****** has two ***** asterisk characters, it is considered to be a **single** matching code.

The ****** pattern is basically the same as the ***** pattern except it will **not** treat a **** as a delimiter which stops the scan.

Why would you use the ****** code in a pattern? Suppose you want a pattern that only applies to file on a particular drive. Let's say you have Linux files stored on drive **L:** and you want to edit some of the text files on that drive. In Linux, it uses just **LF** to end a line, rather than the **CRLF**

in Windows. But, it uses **txt** for text files, the same as Windows. How can we get SPFLite to edit such files with the right End-of-line setting, but **not** change the names of any Linux text files?

Let's create the EFT rule for this, and then we will explain it. Here's the rule:

```
"L:\**\*.txt" = TXT_LF
```

First, a note about Linux file names. If you are on Windows, and were to see a name like **L:\path\data\ten.txt** that is what **Windows** is showing you as the name – but that's **not** its real name, which would be something more like **/usr/path/data/ten.txt** instead of a Windows-style name.

When you access a Linux drive via Windows, the network device driver you use will convert the Linux-style name into a Windows-style name. When you refer to a Linux name in an EFT path, you can use Linux / forward slashes if you want (and they will get converted to \ form) but otherwise the file has to appear like Windows uses it.

When the ****** appears in the pattern, it will match any file name characters including the \ path separator. That means the pattern above would match any of these names:

```
L:\one\ten.txt
L:\one\two\ten.txt
L:\one\two\three\ten.txt
```

and so on.

Suppose you had a file like **L:\ten.txt** with just one \ present. How do you match that? Use this:

```
"L:\**.txt" = TXT_LF
```

When you edit any of these files in SPFLite, they will now use the Profile named **TXT_LF**. All that is left for you is to create or update your **TXT_LF** Profile (and be sure to set the **FORMAT End-of-Line to LF**)

If you create a new Profile using the **PROF NEW new-prof** command, the Profile will be created and it will place you into the Profile Edit Dialog for that new Profile. You can then set whatever unique options needed for the new Profile before actually opening a file which needs it.

This introduction is pretty much all you'll require to use EFT for the majority of your needs. If you need assistance, please use the [SPFLite Forum](#), or contact [us](#) for assistance.

External File Changes

Contents of Article

[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**.

These options have the following meaning:

ALL

SPFLite will notify you in all cases when you have a file opened for Edit, View 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 View or 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.

File Lists - FLISTS

Contents of Article

[Creating and manually editing File Lists](#)
[Extended Path Support](#)
[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".

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 most recent 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 (or via [AUTOFAV](#)), and do not automatically drop off the list.
- In addition to the "standard" **Favorite Files** File List, users can create their own **Named** 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, View 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]

Note: The | in the syntax above is used as a field delimiter, **not** as an OR indicator. e.g. an entry with two fields would appear as **Filename|***

where:

File Path/Name	This should be either a fully qualified filename, or, A path/folder name ending in a \. The string may be entered un-quoted or enclosed in double quotes.
-----------------------	---

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.
------------------	--

Other Fields	When other fields are present (if you're editing an FLIST) please do not modify them. They are used internally by SPFLite. You may modify the 1st two fields (above) as you like. If you make <u>significant</u> changes to the 1st two fields, it would be best to delete the remaining
---------------------	--

fields as they no longer reflect what's in the first two fields.

Extended Path Support

When entering a Path name, you have three choices as to how the path is to be handled by File Manager

The Path ends in a single \	The files in the path, and any lower level path <u>names</u> , are read and included in the list.
The Path ends in a double \	The files in the path, and the files in all lower level paths are read and included in the list. i.e. the full path tree is processed.
The Path ends in a triple \	The pathname <u>itself</u> will be added to the list, <u>not</u> it's contents. Selecting this entry from the list will open the selected path in a new display.

Using Wildcards

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 | |ADD| 2021-03-16  12:16
```

- 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. Each entry has embedded in it the date of last reference, and the Mode of file access. (i.e. what SPFLite function performed the access)

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 oldest reference date is dropped off the list. .

When a file already in the **Recent Files** File List is edited again, the reference date associated with the file is updated.

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* 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. Entries in the FLISTS are removed by these 'forget' requests.

Similarly, you may use a [RESET X](#) command to 'undo' previous Exclude request to hide file names.

If you select **FLISTS** 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.

File Manager

Contents of Article

[Example of File Manager display](#)
[File Manager screen layout](#)
[User Specified QuickList Entries](#)
[Managing STANDARD and ALTERNATE Column Layouts](#)
[Directory refresh](#)
[Specifying the criteria for directory displays](#)
[Extended File Pattern Support](#)
[File Manager ALL](#)
[Mouse selection of files](#)
[Right-Click Context Menu](#)
[Changing the file display order](#)
[Managing File Lists](#)
[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.

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:

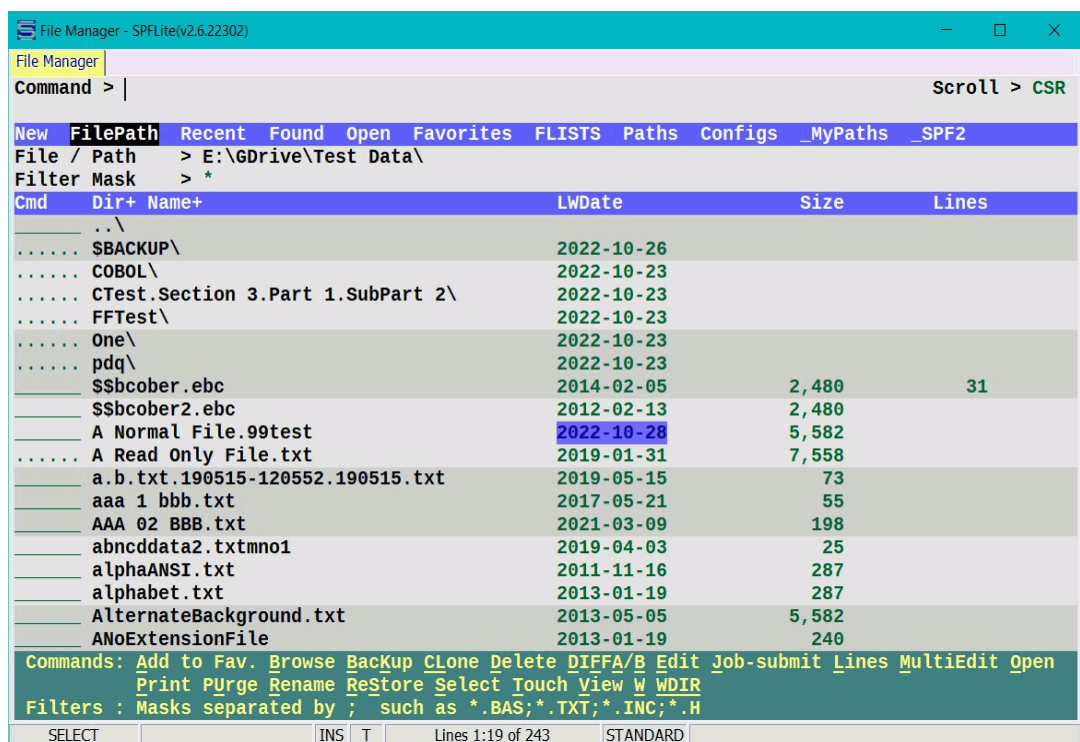
- A variety of predefined File Lists: **Recent Files**, **Favorite Files**, **Open Files**, **Found Lists**, **Recent Paths** and **Config Lists**.
- Directory path lists, both single folder and full recursive path searches.
- Extensive File Pattern matching can be used on the file names to display only 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/View/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.

File Manager can highlight the currently active selection if desired. To do so, go to [Options => Screen](#) and set the BG2 (Background 2) setting for FM Quick Launch Bar to your desired color.

Two Keyboard Primitives are available, (**FMBack**) and (**FMFwd**). These keys allow you to move back and forth between the various FM display screens in a similar manner to moving back and forward in a Web Browser. This is

usually more convenient than using the Quick Launch selections themselves which may also require you to re-enter selection criteria.

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.

Favorites 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.

FLISTS Requests a display of all your saved File Lists.

Paths Requests a display of recently used File Paths.

Config Requests a display of all your current Edit Profiles and Instances.

_SPF2 These two entries are examples of User Specified Quicklist entries which will be described below. It provides quick and simple access to your frequently used FLISTS.

File Path/Name File Patterns

These lines provide the criteria for a Directory folder display. See [Criteria](#) for details.

Column Header

This provides 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 your selected column choices. The specification of what columns and what order you prefer is done in the [Options - File Manager](#) dialog, see that for details.

This portion of the display is scrollable when needed using PgUp/PgDown, Mouse scroll wheel, or keyboard Up/Down keys. If you select more columns than will be visible in the

screen width, they are all created and you may use the normal LEFT/RIGHT commands to scroll the screen left and right. The 1st column (normally the file name) will be "frozen" and will not participate in the left/right scrolling.

If **Hiligh Recent / Active Dates** has been activated, the dates in the LWDate column will be hi-lighted appropriately. See [Hiligh Recent / Active Dates](#) for more information.

User Specified QuickList Entries

Many users have custom FLISTS created which are used frequently. To make access to these quicker, and assuming you have sufficient screen width, you can add up to 6 of your own FLISTS to the right side of the QuickList bar. In the sample screen above you can see SPF23 and SPF24 as examples.

To indicate which of your FLISTS you wish placed on the QuickList bar, simply rename the FLIST to begin with an _ (Underscore). The order will simply be alphabetical from left to right.

Managing STANDARD and ALTERNATE Column Layouts

File Manager allows for a wide range of column data, everyone will have their own preference and the column selection can be set in [Options => File Manager](#). With the addition of additional data in the form of Extended Properties, a single layout simply does not suffice, So there are provisions for two different layouts, one referred to as STANDARD and one as ALTERNATE.

How to switch layouts?

The layout used for any folder path or FLIST can be changed at any time by clicking on the box in the Status Bar that contains name of the current layout in use (STANDARD or ALTERNATE). It will toggle to the other one.

The current (latest) layout in use will be saved by SPFLite and will be used again the next time this folder or FLIST is displayed.

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.

A refresh can also be requested at any time via the (Refresh) keyboard primitive. You obviously must assign this to some key with the [KEYMAP](#) function. This refresh might be needed when the files being displayed were modified by some other non-SPFLite activity.

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.

Recursive requests may be made by ending the pathname with **two ** characters. This

requests a list of the named folder and all nested folders.

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:

Filena TestFile.TXT

me:

Mask: *.TXT;-Test*.*

Result The file would **not** be selected. It is accepted by *.TXT but is rejected by -t: Test*.*. The final overall match result is false.

Filename TestFile.TXT

me:

Mask: *.TXT;-Test*.*;*File.*

Result The file **would** be selected. It is accepted by *.TXT but is rejected by -t: 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 ALL command

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
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. The SELECT line command defaults to a request to EDIT the file

For directories and File Lists, the mouse-select action opens up the directory or File List, and its contents will replace 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.

Right-Click Context Menu

You may Right-Click on any of the list entries to bring up a small Context Menu of commonly used commands. Then you can simply select one of the context menu items with a normal Left-Click.

Note: This will function ONLY IF you do not override the normal (Null) KEYMAP assignment for the Right Mouse Button. The RMB must also have the "Position cursor on mouse-click" selected.

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 any of the displayed column headings. There is also a special heading on the left **Dir**, which controls where folder/directory entries should be placed.

Note: Filenames are normally sorted using the standard Windows Explorer sorting method, where numerical strings are sorted by their logical value, regardless of the length of the numeric string. This option can be overridden by the Options => FM entry "Use simple ANSI sort method". For example of the two sorting results, review the following sample.

Explorer Sort	Simple ANSI Sort
2string	20string
3string	2string
20string	3string
st2ring	st20ring
st3ring	st2ring
st20ring	st3ring
string2	string2
string3	string20
string20	string3

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 an extra character appended to the column heading of the columns which is currently being sorted on. This code can be:

- + the sort is Ascending on this column
- the sort is Descending order on this column
- * the sort is eliminated, the items are in the order they were read from the FLIST or system directory
- . the sort is on File Extension, then File Name (will only be seen on the NAME column.

If you click again on a column which is already selected for sorting, the sort direction will change by rotating through the +, -, *, . options

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 major topic [Working With File Lists](#) provide an extensive description of File Lists and their creation and management.

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.

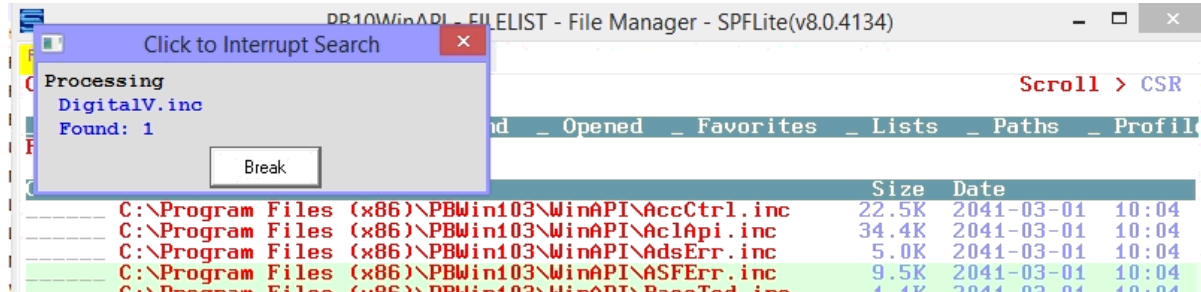
Of course, if you use an **FF** command against the **Found Files** File List with the **same** search string you used to create it, you would just be searching the files you **already found**, and it would **re-find the same files all over again**. That would be called 'going no where fast' (or as a friend used to describe it, "the department of redundancy department").

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**. This allows you to RETRIEVE the FF command and re-use it in an Edit session as a normal FIND 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.

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 CHARS** 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.

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.

<u>ALL</u>	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
<u>BOTTOM</u>	Scroll to the bottom of the directory or File List display.
<u>BROWSE</u>	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.
<u>CD</u>	Change Directory. May only be used if FM is in FilePath mode. It allows a quick change to be made to the File / Path entry. e.g. CD D: or CD MYFOLDER.
<u>CLIP</u>	Open an edit session to directly edit the contents of the Windows Clipboard.
<u>CLONE</u>	Clone a copy of an existing file. You must specify the filename operand when using this command in File Manager
<u>CLS</u>	Clear (or Close) any macro processing Debug screen
<u>CMD</u>	Used to execute an external program or script.
<u>CRETRIEV</u>	Conditionally move the cursor to the Home position or Retrieve the last saved command from the Retrieve Stack.
<u>CSV</u>	This will create a CSV (Comma Separated Value) file containing the current displayed Filelist.
<u>DIFF</u>	Invoke a DIFF file compare function.
<u>DO</u>	Invoke a DO style macro function.
<u>DOWN</u>	Scroll the display down by the amount shown in the Scroll amount field. Typically mapped to the PageDn key.
<u>EDIT</u>	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.
<u>END</u>	Return 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.

<u>EXCLUDE</u>	Exclude files from a File List display based on a mask style operand.
<u>FF</u>	<p>The Find in Files command FF searches all currently displayed file names for a string value. See Performing a file search below.</p> <p>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.</p>
<u>FIND</u>	<p>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'.</p> <p>Do not confuse the File Manger's FIND command with the Edit/Browse command FIND. The two commands are merely spelled the same but are unrelated.</p>
<u>HELP</u>	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.
<u>INSTANCE</u>	Specify a new Instance. The current SPFLite session will be closed, and a new session using the specified Instance will be opened.
<u>KEYMAP</u>	Brings up the SPFLite KEYMAP facility
<u>LOCATE</u>	<p>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.</p> <p>Note that the File Manager LOCATE command is unrelated to the editor LOCATE command. See also the FIND command above.</p>
<u>MAKELIST</u>	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.
<u>MD</u>	<p>Make Directory. Can be used to create a new sub-folder under the existing File / Path.</p> <p>If the operand contains spaces, it must be enclosed in quotes.</p>
<u>MEDIT</u>	Add a file to a MEDIT session
<u>NOTIFY</u>	Setup file modification notice handling
<u>OPEN</u>	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.
<u>OPTIONS</u>	Bring up the SPFLite Global Options window. See OPTIONS - Set Global Editor Options for more information.
<u>PRINT</u>	The PRINT SETUP command may be launched from File Manager or from an

<u>SETUP</u>	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.
<u>PROF EDIT</u> <u>PROF NEW</u>	Bring up the Profile Edit dialog to create/Edit a file Profile
<u>RECALL</u>	Requests opening a specified FLIST
<u>RESET</u>	Resets the hidden status of Excluded and/or Forgotten files in a File List
<u>RFIND</u>	Repeat the previous FIND command
<u>RETF</u>	Retrieves the 'next' saved command from the Retrieve Stack, in a forward direction, opposite to that done by the RETRIEVE command.
<u>RETRIEVE</u>	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.
<u>SET</u>	SET brings up the SET Variable Editor.
<u>SWAP</u>	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.
<u>TOP</u>	Scroll to the top of the directory or File List display.
<u>UP</u>	Scroll the display up by the amount shown in the Scroll amount field. Typically mapped to the Page Up key.
<u>VIEW</u>	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 .
<u>XSUBMIT</u>	Perform SUBMIT processing of a file

Created with the Personal Edition of HelpNDoc: [Say Goodbye to Documentation Headaches with a Help Authoring Tool](#)

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**, **RENAME** and **W** 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

You may select multiple lines in File Manager much as you select multiple lines in a normal Edit screen. All the commands can be used in block mode by simply repeating their last character. For example (**A**, **B**, **C**) would be (**AA,BB,CC**). Longer commands like (**ADD**, **DEL**) would be (**ADDD,DELL**). You can then select multiple lines by using the paired block mode technique (**MM / MM**, **DELL/DELL**, etc) on the first and last lines of a range, or by adding a numeric count to the command. e.g. **D4** would request a Delete on this line, and the next 3 lines.

You may also use the line command modifiers **\ and /** on a command. The **/** (forward slash) requests the command be repeated from the selected line to the bottom of the list. The **** (backward slash) requests a repeat from the selected line to the top of the file.

Note: To support users with other national keyboards where **/** and **** may be awkward to type, the following equivalents may be used.

- . **(period)** to replace **/**
- , **(comma)** to replace ****
- .. **(2 periods)** to replace ****

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
%IMacr-Name	If an IMacro 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 IMacro override. i.e. it is a shortcut for entering EDIT %IMacro-Name
/XForm-Macro	If an XMacro 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 XMacro override. i.e. it is a shortcut for entering EDIT /XMacro-Name
A AA [file-list-name] ADD ADDD	<p>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 default (unnamed) Favorite Files File List. You cannot add directories or other File List names to a File List.</p> <p>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.</p>

	<p>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.</p> <p>There is also available an AUTOFAV facility where files can be automatically added to Favorite lists. See "Using AUTOFAV to create FILISTs".</p>
ALL command	<p>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.</p>
B BB [.Profile-name] BRO BROO BROWSE BROWSEE	<p>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.</p> <p>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.</p>
BACKUP BACKUPP [?] BACK BACKK BK BKK	<p>The Backup command will create a new Backup file for the selected File line. See "Working with BACKUP & RESTORE" for details of Backup file support.</p> <p>Entering BACKUP ? requests a display of the current number of available backups for this file</p>
CANCEL CANCELL CAN CANN	<p>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</p>
C CC	<p>Mark line(s) for use by an FM Primary command, like CUT.</p>
D DD [U] DEL DELL DELETE DELETEE	<p>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.</p> <p>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 <u>Recycled</u>. If unchecked, a popup message will ask for confirmation before the file is Permanently Deleted.</p> <p>You MAY override the normal Recycle Bin choice by adding the optional operand U (unconditional) which will force the delete to be a Permanent Delete.</p> <p>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</p>

	<p>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").</p> <p>Deleting a File List is permitted.</p> <p>A directory can be deleted with the D line command, but only if it is empty.</p> <p>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</p>
DFA DFB	These two commands allow you to select two files and request a DIFF process be started with these files as the default selection.
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 EE [.Profile-name] EDIT EDITT .Profile-name	<p>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.</p> <p>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.</p> <p>File names in a File List which are Guarded will be shown preceded by a character.</p> <p>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.</p> <p>Note that you can use ALL E to Edit all the files listed in a File List.</p> <p>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.</p> <p>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.</p>
END ENDD	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

EX EXX command-name EXEC EXECC	This allows you to invoke a system command (program, BAT file etc) against the filename of the FM line. The system CURDIR will be switched to the folder containing the file before the command is executed. This of course will only work if the command invoked takes a single filename as its command line operand. e.g. EX NOTEPAD would open the file in the normal NOTEPAD utility.
F FF FORGET FORGETT	Forget the file. Applies only to entries created by a File List. The Forget operation removes the file from the File List. Forgetting a file only removes it from a File List; it does not delete the file.
FPROP FPROPP FP FPP	This will bring up the normal System File Properties dialog for the selected file.
J JJ JOB JOBB SUB SUBB SUBMIT SUBMITT	<p>Job Submit. The file is "submitted" using existing user-defined 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.</p> <p>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.</p>
CL CLL [.Profile-name] CLONE CLONEE	<p>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.</p> <p>A directory cannot be cloned with the CL 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.</p>
L LL [U] LINE LINEE LINES LINESS	<p>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.</p> <p>Note: If the current FM data collected already contains a value for the LINES column, it will not process the file again to obtain a 'fresh' value for LINES. If you wish, add the optional U operand (Unconditional) to force a full refresh of the value.</p> <p>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.</p>

M MM MEDIT MEDITT	<p>Multi-Edit. This is the ability to perform a multi-edit session. A <u>multi-edit session</u> 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.</p> <p>Note: you can use ALL M to start a Multi-Edit session using all the files listed in a File List.</p>
N NN NORM NORMM	<p>Normalize will review the contents of an FLIST and remove entries that no longer point at real files or folders.</p>
OPEN OPENN O OO OPENV OPENVV OV OVV OPENB OPENBB OB OBB	<p>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>
P PP PRINT PRINTT	<p>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.</p> <p>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.</p>
R RR [newfilename] REN RENN RENAME RENAMEE	<p>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.</p> <p>When you rename a file, if it is recorded within any current File Lists, the entries within those File Lists will also be renamed.</p> <p>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.</p>
RESTORE RESTOREE RS RSS RESTORET RESTORETT RST RSTT	<p>If RESTORE RS is entered, it requests the file be Restored from this backup to it's original location. If the original file still exists, you will be asked to confirm the overwrite of the file.</p> <p>If RESTORET RST is entered, this command requests a time-stamped version of the file be Restored from this backup to it's original location. This allows you to restore a copy of the file</p>

	<p>without damaging or replacing the existing version.</p> <p>See "Working with BACKUP & RESTORE" for details of Backup file support.</p>
SAVE SAVEE	<p>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</p>
S SS [.Profile-name] SEL SELL SELECT SELECTT	<p>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.</p> <p>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.</p> <p>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.</p> <p>If you click on a file name to invoke the Select function, you can enter the .Profile-name by itself on the File manager line command area, then click on the file name, without having to use the S or SEL command.</p>
T TT TOUCH TOUCHH	<p>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.</p>
V VV [.Profile-name] VIEW VIEWW	<p>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.</p> <p>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.</p>
W WW [Opt-CmdName]	<p>If entered with no operand, the W command will open the file via Windows Shell. i.e. Open the file using its normal default application. But what if you want to process the file with some other command of batch file.</p> <p>If you provide a command name as [Opt-CmdName] it will process the file using this command. This allows you to invoke a system command (program, BAT file etc) against the filename of the FM line.</p> <p>The system CURDIR will be switched to the folder containing the file before the command is executed. This of course will only work if the command invoked takes a single filename as its command line operand. e.g. W NOTEPAD would open the file in the normal</p>

	NOTEPAD utility.
WDIR WDIRR	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.

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

File Profiles

Contents of Article

[The DEFAULT Profile](#)
[What settings are maintained in each Profile ?](#)
[Automatic Colorization linkage](#)
[Managing changes to a Profile](#)
[Profile display](#)
[Profile locking](#)
[Extended File Types](#)
[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 create multiple Profiles, each with different settings to handle various file types. The EFT (Extended File Types) support allows you to flexibly associate the different file types to the Profile best suited for them. This can be as simple as a one-for-one match of file extension to Profile name, or a more complex filtering based on the files location, name, format etc. Review [Extended File Types](#) for a full description of these capabilities.

Profile information is stored in the **SPFLite** CFG file, each Profile is stored independently.

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 not only creates "Profile Clutter", but if you decide to alter one or two Profile options, you are then faced with 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 select an appropriate Profile file, it displays the following popup message:

Note: If the command (EDIT, CREATE etc.) that triggered the detection of the missing Profile was issued by a macro, the popup will not appear. A NEW profile will automatically be created using the actual **running** set of Profile variables.

Create new Profile

No profile currently exists for file-type TXTMNO1
You may choose one of the following options:

DEFAULT	Use DEFAULT Profile for this file-type
DEFAULT ▼	Create Profile TXTMNO1 based on this existing Profile
(Choose) ▼	Assign a current Profile (using EFT) to use for this file-type
Cancel	Cancel the file OPEN
Mark as NONTEXT	Cancel Open and mark TXTMNO1 as NONTEXT in EFT list
Mark as OPENWITH	Cancel Open and mark TXTMNO1 as OPENWITH in EFT list

Here you can choose one of the following ways to handle this missing Profile:

DEFAULT	Add this file extension (in this example TXTMNO1) to the list of extensions in the EFT table which will automatically use the DEFAULT Profile
Create XXX Profile	Create a new, unique Profile for this filetype. The initial settings for the profile will be copied from whichever existing Profile you choose (Note: DEFAULT is established as a starting point. You can alter any of these settings either from within the Edit session which will be started, or later via a PROF EDIT XXX command to open the Profile Editor.
(Choose) Pulldown List	Set this file type to actually use an existing Profile. This will be done via the EFT (Extended Filetype Table) e.g. an entry will be made in the EFT table to cause the chosen Profile to be used for this filetype. For example you might want a filetype of .H to use the same Profile as .CPP
Cancel	Cancel the opening of this file

Mark as NONTEXT	Cancel the open of this file, and add an entry to the EFT table indicating this is a NONTEXT type file. This will cause File Manager to ignore these files in the future.
Mark as OPENWITH	Cancel the open of this file, and add an entry to the EFT table indicating this is a NONTEXT,OPENWITH WINDOWS type of file. This will cause File Manager to open the file with whatever default application it is associated with in Windows.

The **DEFAULT** profile is also used as the profile for the File Manager tab itself. 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:

ACTION	Specify automatic SAVE
AUTOBKUP	Whether to create a backup file.
AUTOCAPS	Whether to use AUTOCAPS support or not.
AUTONAME	Specify an alternate AUTO file name
AUTOSAVE	Whether to automatically save a file at END processing time.
BOM	Set BOM writing ON / OFF
BNDS	The left/right column boundaries for the file.
CAPS	Whether CAPS is to be forced on or not.
CASE	Default case handling for FIND/CHANGE literals.
CHANGE	Whether string changes are handled in Data Shift (DS) or Column Shift (CS) mode.
COLS	Whether to display a fixed COLS line at top of screen.
COLLATE	The code page used for the character set collating sequence
DCB	Specifies the three Settings RECFM , LRECL and EOL together.
EMACRO	The name of a Macro to be invoked immediately prior to a File Save operation.
EOL	Specifies the End-Of-Line delimiters. Deprecated - Use DCB .
HEX	Whether HEX editing mode should be used as a default for the file type.
HILITE	Whether to use automatic colorization support, and FIND / CHANGE highlighting.
IMACRO	The name of a Macro to be invoked immediately following the loading of the file
LRECL	The record length of the file. 0 = unspecified. Deprecated - Use DCB .
MARK	The current column MARK settings.
MASK	The current model MASK line
MINLEN	The minimum logical record length
MODE	Set and switch a tabs Edit and FIND/CHANGE modes
PAGE	Whether to use PAGE mode for FORMAT End-of-Line AUTO or AUTONL files

<u>PRESERVE</u>	Whether to retain trailing blanks on text lines or not.
<u>RECFM</u>	The record format of the file. e.g. Fixed, Variable, Undefined. Deprecated - Use <u>DCB</u> .
<u>SCROLL</u>	The default scroll amount for UP / DOWN / LEFT / RIGHT commands.
<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, MASK 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](#), [MASK](#) and [TABS](#) line commands.

Using The Profile Edit Dialog

The Profile Edit Dialog provides a simple interface to allow altering multiple Profile settings at one time. The command **PROF EDIT profname** will open a new pop-up dialog that allows you to modify nearly all Profile variables in one location.

The dialog which opens has two tabs to provide all the options available.

The first tab (Profile Editor Options):

SPFLite Profile Editor - Profile = TXT

Editor Options **File Data Options**

☐ AUTOBKUP ON

OFF PRESERVE spaces on SAVE

ANSI Encoding used (SOURCE)

☐ BOM Write ON

ANSI Collate handling

OFF PROMPT Automatic file save at END (AUTOSAVE)

NUMBER and NUMTYPE options can only be set using those Primary commands while editing, using this Profile

0 Record length if fixed length records (LRECL)


U File Record Format (RECFM)

CRLF End of line delimiter type (EOL)

0 Import Tabs width

0 Minimum working text line length

DS CHANGE default shift. DS=Data Shift, CS=Column Shift

 Cancel Done

The instructions for each of these settings was provided previously [HERE](#). Refer to that section for links to all the various settings.

Automatic Colorization linkage

If you utilize [Automatic Colorization Files](#), the name of the colorization control file is normally equal to the name of the Profile being used. The [AUTONAME](#) command can be used to alter this default to any other desired AUTO file

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:

```
EDIT - SPFLite(v3.0.24088.Beta) - e:\GDrive\Test Data\testdata.txt
File Manager testdata.txt
Command > Scroll > CSR

=COLS> -----1-----2-----3-----4-----5-----6-----7-----8-----9
=PROF> PROFILE TXT UNLOCKED, ACTION OFF, AUTOBKUP OFF, AUTOCAPS OFF, AUTONAME NONE
=PROF> AUTOSAVE OFF PROMPT, BOM OFF, CAPS OFF, CASE I, CHANGE DS, COLLATE ANSI
=PROF> COLS ON, COMMENTS 1 67 3 99, EMACRO NONE, EOL CRLF, HEX OFF, HILITE FIND AUTO
=PROF> IMACRO NONE, LRECL 0, MACLIB NONE, MARK ON, MINLEN 0, MODE EDIT, NUMTYPE NONE
=PROF> PAGE ON -3, PRESERVE C, RECFM U, SCROLL CSR, SOURCE ANSI, START FIRST
=PROF> STATE ON, SUBARG OFF, SUBCMD OFF, TABS ON, TABBND OFF, XFORM NONE, XTABS 0
=WORD> A-Z a-z 0-9 _ $
=MARK>
=MASK>
=TABS> *
=COLS> -----1-----2-----3-----4-----5-----6-----7-----8-----9
=BND> <+
000001 '----- INCLUDE command
000002 if ClrInclude(i).Len1 = 0 then
000003 aaa,bbb,ccc,ddd,ebee,fff,ggg,hhh,iii,aaa,bbb,ccc,ddd,eee,fff,ggg,hhh,iii,aaa,bbb,ccc,ddd,e
000004 four(five)six
```

This display shows you the current values for all profile settings.

If the running edit session was invoked using an EFT Override, the display will alter to show the EFT statement used and to hilite the individual values by surrounding them with <...> brackets.

```
EDIT - SPFLite(v3.0.24020) - D:\DOCUMENTS\SPFLITE2\SOURCE\LCmd.inc
File Manager LCmd.inc
Command > | Scroll > CSR

==COLS> ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8---+---9---+---1
*****
***** Top of Data *****
==EFT> .inc = BAS
==PROF> PROFILE BAS UNLOCKED, ACTION OFF, AUTOBKUP OFF, AUTOCAPS ON, AUTONAME BAS
==PROF> AUTOSAVE OFF PROMPT, BOM OFF, CAPS OFF, CASE T, CHANGE DS, COLLATE ANSI
==PROF> COLS ON, COMMENTS 2 67 3 100, EMACRO NONE, EOL CRLF, FOLD OFF, HEX OFF, HILITE FIND AUTO
==PROF> IMACRO NONE, LRECL 0, MACLIB NONE, MARK ON, MINLEN 0, MODE EDIT, NUMTYPE NONE
==PROF> PAGE OFF, PRESERVE OFF, RECFM U, SCROLL CSR, SOURCE ANSI, START FIRST
==PROF> STATE ON, SUBARG OFF, SUBCMD OFF, TABS ON, TABBND OFF, XFORM NONE, XTABS 0
==WORD> A-2 a-z 0-9 _
==MARK>
==MASK>
==TABS>
==COLS> ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---+---8---+---9---+---1
==BNDS>
0000001  '-----+
0000002  ' License Stuff
0000003  '-----+
0000004  '-----+
|
```

Profile locking

If you look at the first =PROF> line in the 1st example 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**.

The 2nd example shows **EFT LOCKED**. This indicates that regardless of the Profile's normal LOCK status, the Profile has been locked since it was temporarily modified by the EFT override support. This is to prevent these temporary overrides being saved back to the normal Profile values.

Extended File Types

You may typically 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 provides a tool that allows you to specify that these file should all use a common Profile. This support is referred to as **Extended File Types**. It basically allows you to **point** a specific filetype to any existing Profile. 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, edit the **EFT** table (accessed by entering an **EFT** command), and add an entry that says **.H = C**

Now, whenever you edit a **.H** filetype, it will use the **.C** Profile

You can of course point multiple filetypes to the same common Profile, there is no limit to the number.

The Opposite Need

There is also a sort of **reverse** requirement when it comes to assigning a Profile to a filetype. Many times you may have multiple files, of widely varying usages, and all end in the same filetype. A good example of this is **.TXT**. Many files end in **.TXT** but actually require significantly different Profiles.

For example, some **.TXT** files may come from external locations and may need to specify End-of-Line as EOL = LF instead of CRLF. Or may need different SOURCE values (ANSI / EBCDIC) etc.

But they all end in **.TXT** ! What to do?

EFT provides extensive masking abilities to allow you to distinguish the variety of filetypes based on other parts of the filename, the folder or drive location. And then to specify another Profile and optionally override 1 or more individual options.

A simple form might be that files needing EOL = LF might always be named with a filename ending in LF, and the following two EFT statements would handle it:

```
\*LF.TXT = TXT,EOL LF
\*.TXT   = TXT
```

In this manner we can handle two unique circumstances with only one Profile.

Refer to "[Working With Extended Filetypes](#)" for full details of **EFT** support, examples of syntax and further practical examples.

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.

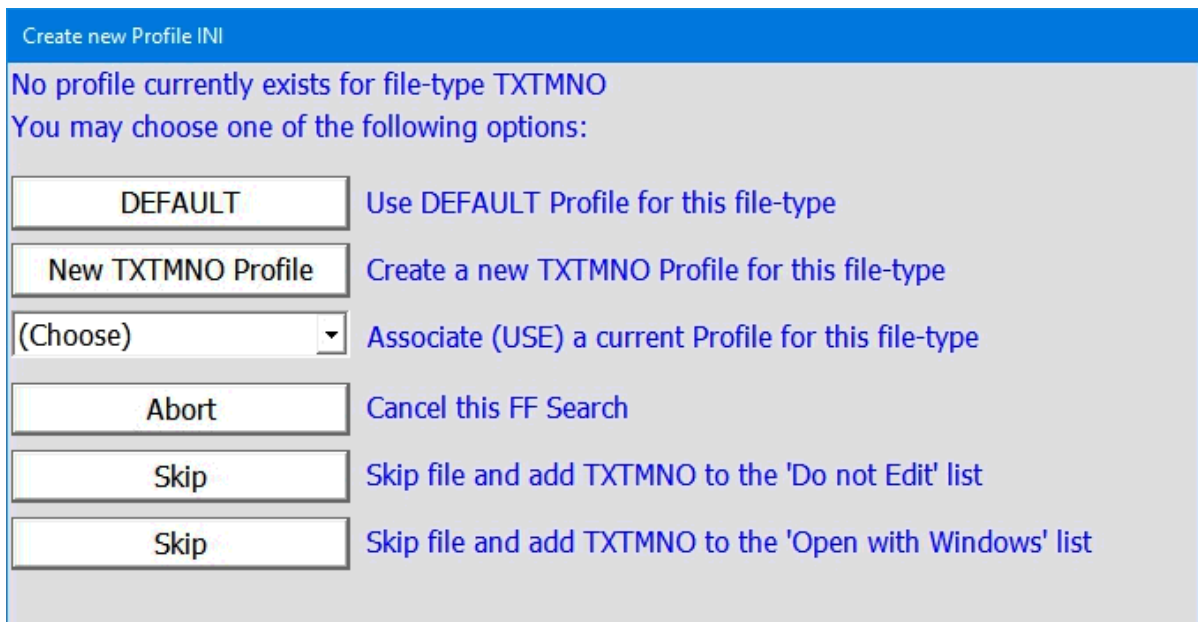
When you **COPY** another profile, the settings of the two profiles are completely independent of each other; a subsequent modification of one of the profiles has no effect on the other profile.

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 [Extended File Types](#))

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 .TXTMNO file type. At this point you can choose one of the 6 options.

- Use the DEFAULT profile for this and any other .TXTMNO files seen subsequently.
- Create a new TXTMNO profile for this file type. It will be based on the DEFAULT

profile.

- Select another existing Profile to be used for .TXTMNO files.
- Abort the FF search at this point.
- Skip this .TXTMNO file, and add it to the EFT NONTEXT list for future searches.
- Skip this .TXTMNO file, and add it to the EFT NONTEXT,OPENWITH WINDOWS list for future searches.

FileOpen Startup Files

The command line operands for SPFLite include an -FILEOPEN operand. This allows you to point at a file containing a list of filenames which you wish to be opened immediately upon startup. The list can contain files to be opened in individual tabs as well as lists of files which are to be opened in a MEdit session ([Multi-File Edit](#))

This section describes the syntax and format of that file. Basically, the whole file is treated as one single logical line. But creating or editing long lists of files in one single line is impractical. So the file can be entered in as many multiple lines as you like.

But keep in mind, these multiple lines will be 'flowed' together into 1 line before being processed. One result is - NO COMMENTS!

The FILEOPEN file also interacts with the normal SPFLite Most Recent Files list (MRF). The MRF list is created when SPFLite is terminated while Edit tabs are open, and is designed to re-open those files again on the next startup.

With the introduction of -FILEOPEN, we have to specify how the FILEOPEN file-list and the MRF file-list are to be handled together.

Syntax for that FILEOPEN single logical line is basically a list of filenames, separated by commas.

Interaction Between FILEOPEN and the MRF List

The FILEOPEN data is handled in one of three ways:

- If a comma is added **before** the list of files, it requests the FILEOPEN be appended to the MRF list.
- If a comma is added **after** the list of files, it requests the reverse, the MRF list be appended to the FILEOPEN.
- If no leading or trailing comma, it requests the FILEOPEN **replace** the MRF list.

Simple Open a File in a separate tab

The File Entry is just the full filename including Path (quotes are not needed)
If multiple filenames, separate them using a comma.

The file will be opened in Edit, If you wish, the name can be prefixed with (V) or (B) to request it be opened in View or Browse. (E) for Edit is also available but is assumed.

Open Multiple Files in a MEdit session

To request a MEdit session the normal simple filename can itself be a list of files, separated by the "|" character. No leading (E) is required, MEdit is always considered Edit mode.

Examples

D:\Documents\FileA.txt,D:\Documents\FileB.txt

Open 2 files in separate tabs, ignore other file re-opens

D:\Documents\FileA.txt|D:\Documents\FileB.txt

Open same 2 files in a single MEdit tab, ignore other MRF file re-opens

(B)D:\Documents\FileA.txt,(V)D:\Documents\FileB.txt

Open the files, one in Browse, one in View, ignore other MRF file re-opens

,D:\Documents\FileA.txt,D:\Documents\FileB.txt

Open 2 files AFTER any other MRF file re-opens

D:\Documents\FileA.txt,D:\Documents\FileB.txt,

Open 2 files BEFORE any other MRF file re-opens

D:\Documents\FileA.txt

,

D:\Documents\FileB.txt

,

Same as tthe previous showing how it COULD be entered on multiple lines.

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Process with HelpNDoc's Project Analyzer](#)

Finding and Changing Data

Contents of Article

[Specifying the Search String](#)

[Simple string](#)

[Quoted String](#)

[Picture String and Format String](#)

[Delimited Search String](#)

[Regular Expression String](#)

[Hexadecimal String](#)

[Character and Text Strings](#)

[Effect of CHANGE/ALIGN 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

Many of the SPFLite commands allow you to specify search criteria to 'narrow down' the lines which are to be acted on by the command. Some common ones are [FIND](#), [EXCLUDE](#) and [CHANGE](#). The search criteria provide a powerful means of selecting the text lines you wish to manipulate. This provides for powerful editing functions because commands can operate on a complete file rather than on a single line.

Specifying the Search String

The primary control for any search is the search string, because it represents the value for which you are looking. T

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 duplicates any of the other valid keyword operands must be quoted. Sometimes it is easier to quote all strings to ensure they are not confused with command keywords.

Quoted String

Any string enclosed 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](#).

Delimited Search String

Like Picture Strings, Delimiter search strings provide another alternative to perform pattern-matching which can be simpler to use than Regular expressions. This literal type allows you to search for strings of varying length which are delimited by some identifiable characters. For example strings within brackets or parenthesis. Or quoted strings. See [Specifying a Delimited Literal](#) for more details

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](#).

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 great than 9 can be in upper or lower case.

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 any of the command's valid keywords. 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/ALIGN 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 for both the [CHANGE](#) and [ALIGN](#) commands.

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](#) or [ALIGN](#) 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. Similarly when shifting data with the [ALIGN](#) command.

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](#) / [ALIGN](#) 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 **MODE DS** or **MODE CS** command.

As well, the shift mode for an individual [CHANGE](#) or [ALIGN](#) command can be explicitly specified by adding a [CS](#) or [DS](#) operand to the 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 position 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 all 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 the string, 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 the found line is always the 1st line of the screen.

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 the string by using the operands **PREFIX**, **SUFFIX**, **WORD**, **CHARS** or the letters **C**, **Q** and **T**. The search context defines "where" or "under what circumstances" a given search string is considered to be "found".

CHARS

The string 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.

PREFIX

Locates the string at the beginning of a word. The string 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 the string at the end of a word. The string 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.

NOTE: Active source colorization is required to utilize the next 3 options. This means [HILITE AUTO ON](#) must be specified in the file Profile, **and** a valid [AUTO](#) file containing colorization information must be present,

C

Specifying **C** requests that the string can only be *found* if it occurs within a defined [COMMENT](#) string.

Q

Specifying **Q** requests that the string can only be *found* if it occurs within a defined [QUOTED](#) string.

T

Specifying **T** requests that the string can only be *found* if it occurs *outside* a defined comment string.

The above three options can also be combined to create custom combinations. e.g. **C T** would request string-1 be found **either** in a COMMENT string, **or** in an open text area (neither quoted or commented)

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

```
FIND 'DO'          - DO
  DONT ADO ADOPT 'DO' (DONT)    Finds all of them

FIND 'DO' CHARS    - DO DONT ADO ADOPT 'DO' (DONT)    SAME THING
- Finds all of them
```

```

FIND PREFIX 'DO'      - DO DONT ADO ADOPT 'DO' (DONT)      Finds DONT
(DONT)

FIND SUFFIX 'DO'      - DO DONT ADO ADOPT 'DO' (DONT)      Finds ADO

FIND WORD 'DO'        - DO DONT ADO ADOPT 'DO' (DONT)      Finds DO an
d 'DO'

FIND 'DO' Q           - DO DONT ADO ADOPTDO' (DONT)        Finds 'DO'

FIND 'DO' C           - DO DONT ADO ADOPT 'DO' (DONT)        Finds none,
there is no

                                                                    defined
COMMENT string

```

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.

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 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 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** 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.

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 - Highlights"](#) 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 (DEFAULT)

General FM Submit Screen KBD Status Schemes Hilights Config

<input checked="" type="checkbox"/> Audible BEEP on Errors	<input checked="" type="checkbox"/> Only 1 SPFLite running
<input checked="" type="checkbox"/> Visual BEEP on Errors	<input checked="" type="checkbox"/> Re-Open last file(s) at start
<input checked="" type="checkbox"/> Delete to Recycle Bin	<input checked="" type="checkbox"/> Warn on modified View file
<input type="checkbox"/> Use WORD as default for FIND/CHANGE commands	<input type="checkbox"/> Display splash screen on startup
<input checked="" type="checkbox"/> Only English A-Z and a-z are considered alphabeti	<input type="checkbox"/> Warn on Open of Non-Text files
<input type="checkbox"/> Default RESET will Revert User line status	<input type="checkbox"/> CUT should default to NEW
<input type="checkbox"/> Open HELP screens in Full Screen mode	<input checked="" type="checkbox"/> Maintain screen position after Line Cmds
<input checked="" type="checkbox"/> Display Macroname on Message line	<input type="checkbox"/> Minimize SPFLite to the System Tray

BACKUP Retention Criteria

RETPD [Days] Minimum Gens [Num] Maximum Gens [Num]

Minimum command length for RETRIEVE

Command separator character

Default # cols to use for data shifts (Min 1) (Append 'P' to use Profile XTAB)

Number of columns/lines per Auto-Scroll

Notify tabs on external file change


Line Commands repeat limit (0=no limit)

Check for SPFLite update interval

Number of UNDO levels (0 to turn off UNDO)

Enter the invalid characters for P'. ' picture literals

Display Character to use for Invalid characters (Or N to suppress translation)

 SPFLite.CFG file Folder:

If you want to quickly change the search context from the edit command line without using the Global Options window, you can issue a **MODE WORDS** or **MODE 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.

Find in Files

Contents of Article

[Syntax](#)

[Example FF Commands](#)

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.

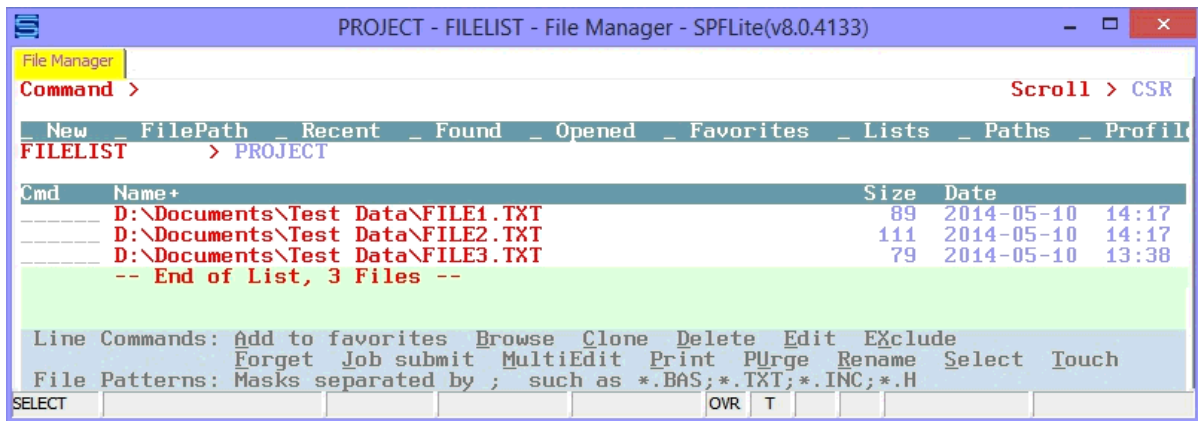
If not specified, a **CHARS** search is done if the status line shows **C** or **T**, and a **WORD** search is done if the status line shows **C W** or **T W**. These two defaults can be set by the [FIND CHARS](#) and [FIND WORDS](#) commands, as in an edit tab.

NF

If the **NF** option is used, a search is made for files that do **not** have the *string* present on any line.

Example FF Commands

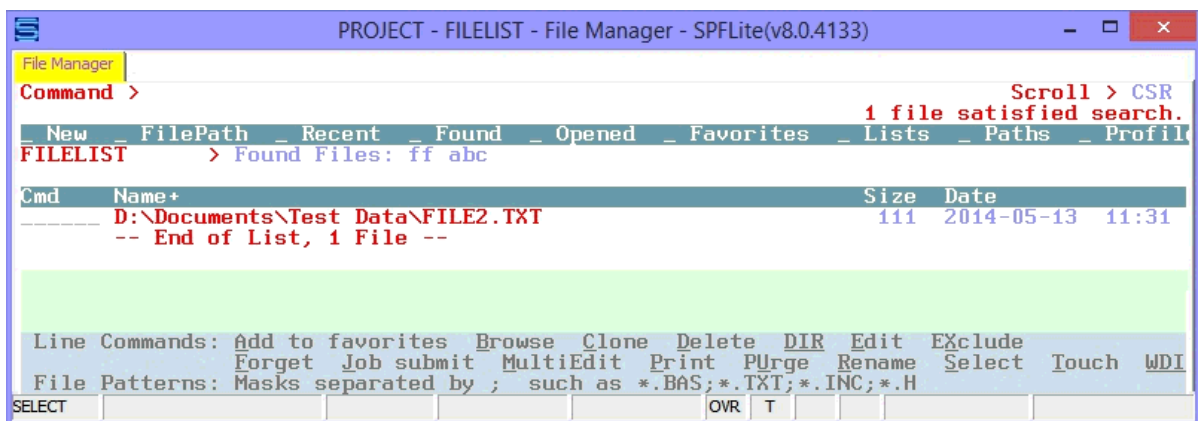
Let's say you have three files in a **Named Favorites** File List called PROJECT, and you want to know which files have the string ABC in them. Click on the File Manager tab and then on the line with **PROJECT** File List. You will see a display something like this:



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:



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.

Including 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.

Instances and Configuration

Introduction

SPFLite, when installed, defaults to a fairly normal allocation of files. A new folder "SPFLite" is created in the user's normal "Documents" folder, and all SPFLite related files are created within that folder. For the vast majority of users this is the simplest and easiest to manage choice.

However SPFLite does provide additional flexibility for more advanced users or those with unique requirements for storage of SPFLite data.

The following section is therefore for **advanced users only**.

As well as providing considerable flexibility in how you can manage the storage of SPFLite parameters and preferences, additional support has been added to assist in running multiple Instances of SPFLite, each with perhaps different sets of preferences and defaults.

Several examples follow to address the more common requirements:

[I'd like to get the SPFLite folder out of \User\Documents](#)

[I'd like to share my SPFLite configuration between systems](#)

[I'd like to create alternate 'Instances' of SPFLite and customize each of them](#)

[Using the INSTANCE command to control Instances](#)

I'd like to get the SPFLite folder out of \User\Documents

If you'd like to move the storage of SPFLite configuration data out of \User\Documents\SPFLite\ to another location, then here is the procedure.

1. Ensure all copies of SPFLite2.EXE are not running.
2. COPY (**not MOVE**) the \User\Documents\SPFLite\ folder to your new desired location.
3. Start SPFLite2.EXE and enter OPTIONS CONFIG.
4. At the bottom of the Config tab, alter the two location boxes to your new folder location(s).
5. Click DONE, you will be reminded to Restart SPFLite.
6. Shut down and Restart SPFLite2.EXE, you will be running on the Moved config files.
7. Clean up the old SPFLite config folders at your leisure.

I'd like to share my SPFLite configuration between systems

This need is basically the same as that above. Just choose a Network drive which can be accessed from all involved systems.

Do the change on one system using the instructions above.

Then do the same on each of the other systems, omitting step 2 (Since the data is already

ON the Network drive)

I'd like to create alternate 'Instances' of SPFLite and customize each of them

SPFLite support will now allow creation of multiple 'versions' of the execution environment. These versions are referred to as "Instances". Each Instance can:

1. Have a unique set of preferences. Preferences are all those settings accessed via the [OPTIONS](#) command.
2. Share the DEFAULT EFT table, or have a private, unique EFT table.
1. Share the DEFAULT keyboard definition, or have a private, unique keyboard definition.
3. Share the DEFAULT SET variables, or have a private, unique collection of SET variables.
4. Share the DEFAULT command RETRIEVE history, or maintain its own command history.
5. Be identified uniquely in the window title bar. The Instance name will replace the normal SPFLite version number.
6. If standard windows shortcuts are used to invoke the different Instances, SPFLite now ships with some additional (6) different Icons which can be used to customize the shortcuts for easier reference.



SPFLiteBlackTransparent.ICO



SPFLiteWhiteTransparent.ICO



SPFLiteYellow1.ICO



SPFLiteYellow2.ICO



SPFLiteBlue1.ICO

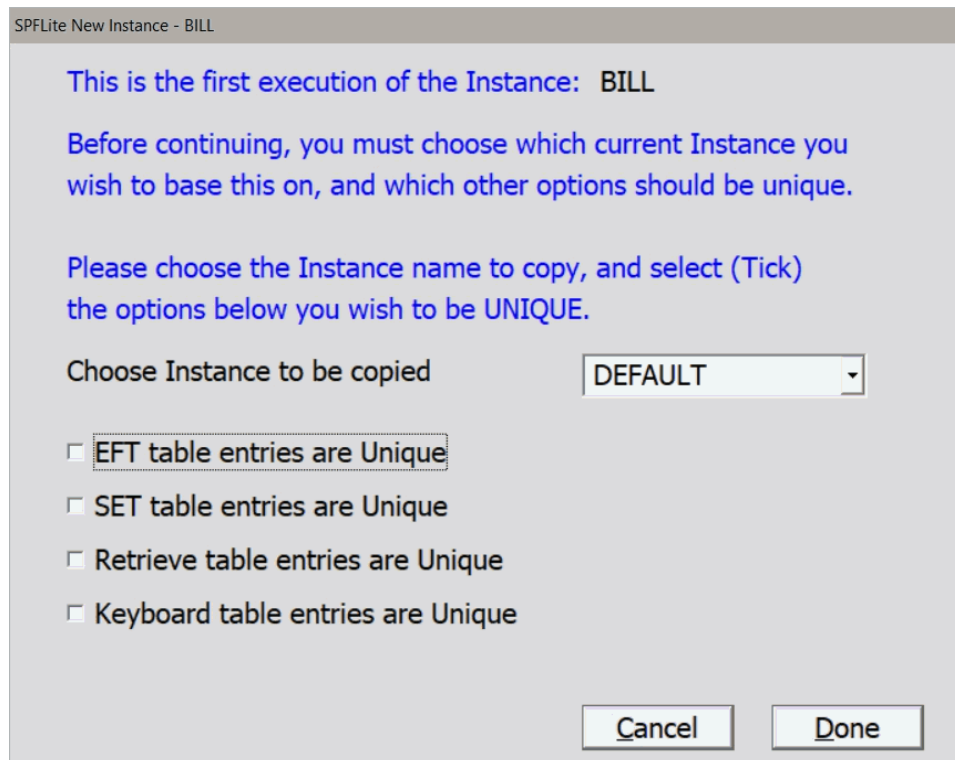


SPFLiteBlue2.ICO

Setting up a new Instance

To create a new Instance: say it is named BILL)

1. Copy an existing SPFLite shortcut and rename it to identify the Instance.
2. Open Properties for the shortcut.
3. Edit the Target command in the Windows Shortcut and add -INSTANCE BILL (or just -I BILL)
4. If desired at this point, change the Icon to one of the alternate ones provided. The Icons are located in the normal install folder where the EXE file was placed.
5. Close the Properties.
6. Double click the Icon.
7. You will see the following Prompt:



1. Choose the name of the current Instance from the drop-down menu which is to be copied as the initial settings for the new BILL instance.
2. Select the Unique options. When selected (Ticked) it indicates you wish to make these items unique for the new Instance. If unticked, it means you want this new Instance to share the DEFAULT versions.
3. Select DONE. The Instance will be created and SPFLite will continue initialization. The EFT, SET, KBD and Retrieve tables will be copied from the Instance name you chose in Step 1
4. The Instance can later be customized as you desire using OPTIONS, KEYMAP, etc.

How to Start an existing SPFLite Instance

To start an existing Instance (say it is named FRED), simply add the following to the Target command in the Windows Shortcut.

-INSTANCE FRED (or just -I FRED)

How to Remove (Delete) an existing, no longer required Instance

To delete an existing Instance, simply go to File Manager, select the Config Display (on the right) and enter a "D" next to the Instance you wish to delete.

Identifying What Instance is Running

The currently running Instance will always be identified in the Windows Title bar. As well, popup dialogues (like Options, KEYMAP, etc.) will always show the Instance name in the title bar. This is important since an Instance may be sharing some preferences (like Keyboard) with the DEFAULT Instance and even though you are running a separate Instance, changing the Keyboard definitions may be altering the DEFAULT keyboard definition. Always check what is displayed if you are not sure. If you use the "Minimize to the System Tray" option, hovering over the tray Icon will display the Instance name for all non-DEFAULT sessions.

Using the INSTANCE command to control Instances

Although Instances can be invoked with a customized Icon as described above, if you have more than a couple instances, this becomes more and more impractical. The **INSTANCE** command allows you to startup and/or switch to another instance quickly, without the need to customize an Icon, or open a command window to add additional operands to the startup.

How to create a new Instance

Let's assume you are running a normal Default SPFLite session, and you want to create one named TEST.

Enter:

INSTANCE TEST KEEP (INSTANCE can also be abbreviated as INST)

The **KEEP** operand requests the current DEFAULT session remain active. A new SPFLite windows will appear as described in "Setting Up a New Instance" above. Follow those instructions and your new Instance will be active.

If you don't require the existing DEFAULT session to remain open leave KEEP off. If you have open Edit tabs, use the SAVE or CANCEL operands to indicate how you want them handled.

How to switch to another existing Instance

This is identical to creating a new instance, except the creation step is bypassed.

Using the -F and -D operands

When switching to another instance, you may request the addition of the **-FILEOPEN** and **-DO** startup operands.

Enter:

INST TEST -F C:\MYLIST.TXT -D MYDOMACRO CAN

This requests a switch to a TEST Instance, adding the -FILEOPEN and -DO operands. The existing session will close and any open edit tabs will be closed with a CANCEL command.

Switching to the existing Instance

Yes, this sounds a bit weird, but it does have a purpose. Using the **-F** operand, it allows you to use the FILEOPEN support to open a list of filenames in the current session. Assuming you are running in the DEFAULT Instance, and have created a FILEOPEN file named "FILES.TXT".

Enter:

INST DEFAULT KEEP -F C:\ADDFILE.TXT

This will result in the files listed in the ADDFILE.TXT file to be opened in the current DEFAULT session.

Switching Instances and keeping the active tabs

Say you have several files open in the DEFAULT Instance, but you want to continue editing them in your FRENCH Instance (which has, say, unique keyboard mapping). As long as SAVEing them during the switch is OK, you can.

Enter:

INST FRENCH -F * END

This uses the feature of INSTANCE which saves the current tabs file list as _FILEOPEN.TXT in the SPFLite Home folder. The "*" operand for **-F** requests this standard file be used as the FILEOPEN. The current session will close all tabs via END, and then the same files will be re-opened in the new FRENCH instance.

Password Protection

Line Labels

Contents of Article

[Relative line label notation](#)

[Temporary line labels, also known as line-number pseudo labels](#)

[Using the single character line range arguments](#)

[Using Labels with the LINE Command](#)

[Using Labels as Bookmarks](#)

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 IBM's 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

.ZFIND	to indicate the last (most recent) line found by a FIND command
.ZFINDA	to indicate the <u>First</u> line found in a FIND ALL request
.ZFINDB	to indicate the <u>Last</u> line found in a FIND ALL request
.ZLOC	to indicate the last (most recent) line found by a LOCATE command
.ZLOCA	to indicate the <u>First</u> line found by a LOCATE ALL command
.ZLOCB	to indicate the <u>Last</u> line found by a LOCATE ALL 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.

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

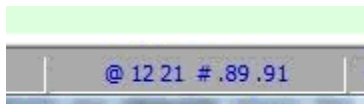
Suppose you *knew* ahead of time that the .A label will go on line 100, and the .B label will go on line 200. With SPFLite you no longer have to "invent" these .A and .B labels. You can use *temporary line labels*, also known as *line-number pseudo labels*. Such a pseudo label is a . dot followed by 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.

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.

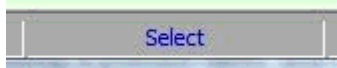
For example, using the above displayed sample, a command line of **PRINT #** would be treated as if it were entered as **PRINT .89 .91** to print the selected line range.

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 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</i>
<code>LINE ' . '</code>	
<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>

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.

Using Labels as Bookmarks

Labels are obviously well suited to use as Bookmarks. A discussion and examples can be found [here](#).

Line Lengths

Contents of Article

[LRECL Considerations](#)

[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 FORMAT End-of-Line 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.

LRECL Considerations

When setting LRECL to a value greater than 0, you need to be mindful of what this value means and how it is processed by SPFLite.

When a file is opened that is defined with Record-length N (where N is > 0) and the file has Record-Format F and End-of-Line NONE, SPFLite uses the N value to 'divide up' the file data into 'pieces' that are exactly N characters each. Here, the N value is needed because otherwise there would be no way to know how long each "piece" is. On IBM mainframes, they use the LRECL value for exactly the same reason.

However, once your data is in the editor, SPFLite performs its normal function as a text editor, where "text" lines can vary to any arbitrary size. SPFLite does NOT prevent any of your text lines from exceeding a length of N characters.

Why doesn't it do that? First, it would be difficult to implement, since it would mean SPFLite would have to operate in two different 'modes' - where one only allowed fixed-length records and the other allowed line sizes to vary. Second, you might want to place information 'outside' of the LRECL size, perhaps as "notes" or as temporary information, which you intended to truncate or discard later. You are allowed the flexibility to do such things.

What happens if your edit session has created lines longer than N characters? While you are editing, nothing happens. But when you go to SAVE the file, or END the edit when AUTOSAVE is in effect, SPFLite will issue a warning message like this:

*File contains lines longer than Record-Length of nnn
Click YES to continue and truncate lines, or
Click NO to abandon the file save and return to Edit*

If you didn't intend to create "long" lines you may want to investigate what has happened. Reply "NO" and perhaps do a **FIND ALL P'↵' n n+20** (where $n = \text{LRECL} + 1$) to find the extra data. If you want to truncate all of your data to a fixed length, you can use the TL/TLL (Truncate to Length) line command. Or click on YES and simply discard data past the LRECL value

Other ways to truncate your data to a given length are:

- Enter Power Typing mode on ALL lines, position the cursor to one column to the right of the last valid column, and press the (EraseEOL) key, usually assigned to the ESC key.
- Exclude all lines of the file except for the first and last. Then, highlight a 'block' of characters, where the left side of the 'block' is one column to the right of the last valid column, and press the DEL key.

The MINLEN profile option

To define a "garden variety" standard, variable-length Windows text file, the associated PROFILE settings are **FORMAT U CRLF 0**. In SPFLite, this term really means there is **no maximum** logical record length. For files of type Record-format is **F**, the Record-Length must be a positive number defining the **fixed** line size.

In addition to the Record-Length 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.

NOTE: **MINLEN** affects the contents of the file "while you are editing", it does NOT necessarily affect the file when it is written. See the next section and the Profile option [PRESERVE](#) for details.

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.

PRESERVE handling

PRESERVE specifies how trailing blanks on a line are to be handled, so it does interact with **MINLEN**.

The extra blanks that may be added by **MINLEN** processing are not treated specially. So if you specify **PRESERVE ON**, these extra added blanks will be written to the file. So don't specify an unnecessarily large value for **MINLEN** if you use **PRESERVE ON**, you could add considerably to your file size.

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.

Line Tags

[Why use tagged lines ?](#)

[Example: Changing tagged lines](#)

[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](#)

[Using 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 also 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), even if it doesn't "fit". 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.

Like labels, 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 the first position (the **:** colon), 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 a **:tag** 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+

```

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
***** Top of Data *****
000001 ABC
:T ABC
000003 ABC
:T ABC
000005 ABC
:T ABC
***** 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. The option STATE is "ON" when you see **S+** on the status bar.

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 (sometimes referred to as "pop out line") when strings 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 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, and the syntax is more involved. 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.

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 sequence area field 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*.

Note: Sometimes we call that usually-blank column is called the **gap column**. As with character positions in the sequence area, the gap column has no column number, since it's not data.

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:

```

***** Top of Data *****
000001 ABC
:T ABC
000003 .AA : ABC
000005 ABC
:T ABC
***** Bottom of Data *****

```

File Manager: Tags.TXT

Command >

Scroll > HALF

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. If you put the cursor on the A of line 6, the status bar will just show **L 000006 - 6**.

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:

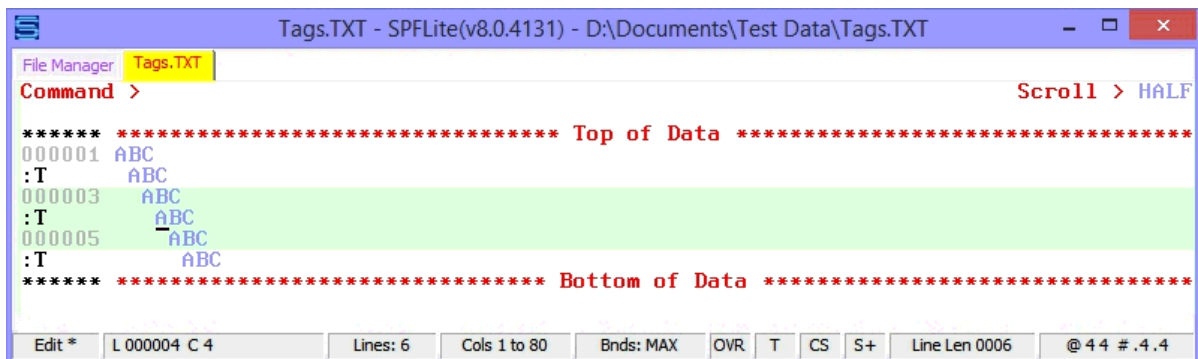
Alt-period = { . }

Alt-semicolon = { : }

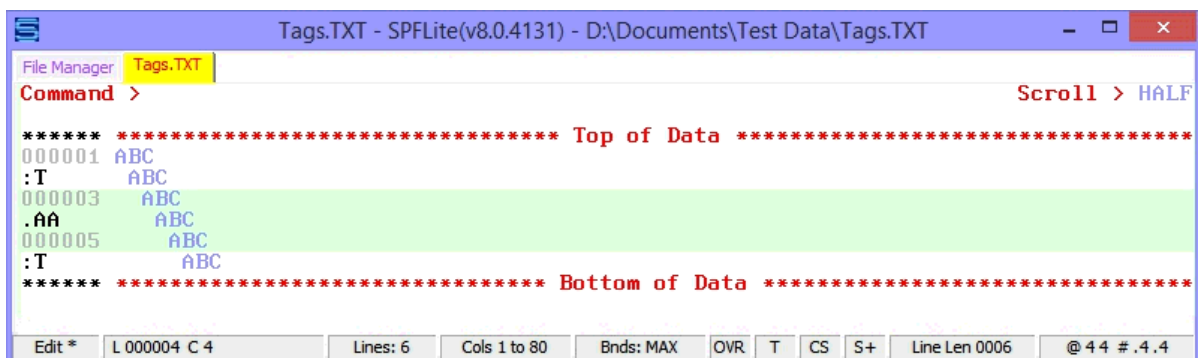
Note: These are not defaults; you would have to do this key mapping yourself.

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 in the *gap column* would go away, the tag would reappear, and line 4 would look like this:



Whereas, if you placed the cursor on line 4, and were to press **Alt-semicolon**, the *tag* and colon would go away, and the line 4 would look like this:



In both cases, once the label/tag co-location situation 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** | **SET**]

[**FIRST** | **LAST** | **NEXT** | **PREV** | **ALL**]

[**PREFIX** | **SUFFIX** | **WORD** | **CHAR**]

[*line-control-range*]

[**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** 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*, which it's not.

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 here.

- The *search-string* is used to **find text** on particular lines for which you wish to set, clear 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. **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**, **D**, **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**.)
- 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** 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 **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.
- 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. ***That is, changing whether a line has a tag - or not - is a "modification" of the line.*** 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” – will be left alone.

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, a line tag, an **X** or **NX** excluded-line selection or a **U** or **NU** user 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 *new* tag 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.

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

```

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.

Created with the Personal Edition of HelpNDoc: [Protect Your Confidential PDFs with These Simple Security Measures](#)

LINE Primary Command

Contents of Article

[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, 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](#).

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**" 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. These dual-use commands are:

Line command	Primary command
D / DD	DELETE
LC / LCC	LC
S	SHOW
SC / SCC	SC
TC / TCC	TC
UC / UCC	UC
X	EXCLUDE

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** primary command 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 this 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 could be mistaken for an actual operand of **LINE** itself, it must be enclosed in quotes. 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: 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** primary command 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 forum 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.

Result:

Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT

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.

Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT

File Manager Line-Cmd.TXT

Command > LINE R .4 .6

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:

Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT

File Manager Line-Cmd.TXT

Command >

Scroll > HALF
'R' applied 3 times

***** Top of Data *****

```

000001 one
000002 two
000003 three
000004 FOUR
000005 FOUR
000006 FIVE
000007 FIVE
000008 SIX
000009 SIX

```

***** Bottom of Data *****

Edit * L 000005 Lines: 9 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0008

LINE Example 3: Shifting tagged lines

LINE] :M ALL is used to shift all lines with line tag **:M** on them.

Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT

File Manager Line-Cmd.TXT

Command > LINE J :M ALL Scroll > HALF

***** Top of Data *****

000001 one
:M two
000003 three
:M four
000005 five
:M six

***** Bottom of Data *****

Edit * 2014-05-13 12:09:09 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result:

Line-Cmd.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Line-Cmd.TXT

File Manager Line-Cmd.TXT

Command > Scroll > HALF

***** Top of Data *****

000001 one
:M two
000003 three
:M four
000005 five
:M six

***** Bottom of Data *****

Edit * L 000002 C 1 Lines: 6 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

MACROS

SPFLite provides a full macro language that can be used to automate or extend the normal built-in functions. The MACRO facility allows creation of new Primary commands as well as new Line Commands. It also supports functions in the File Manager environment.

Details are provided in this [Macros](#) section of this Help document.

In an SPFLite session enter **HELP MACROS** on the command line to open Help at the Macros section of this document.

MultiEdit Sessions

Contents of Article

[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).

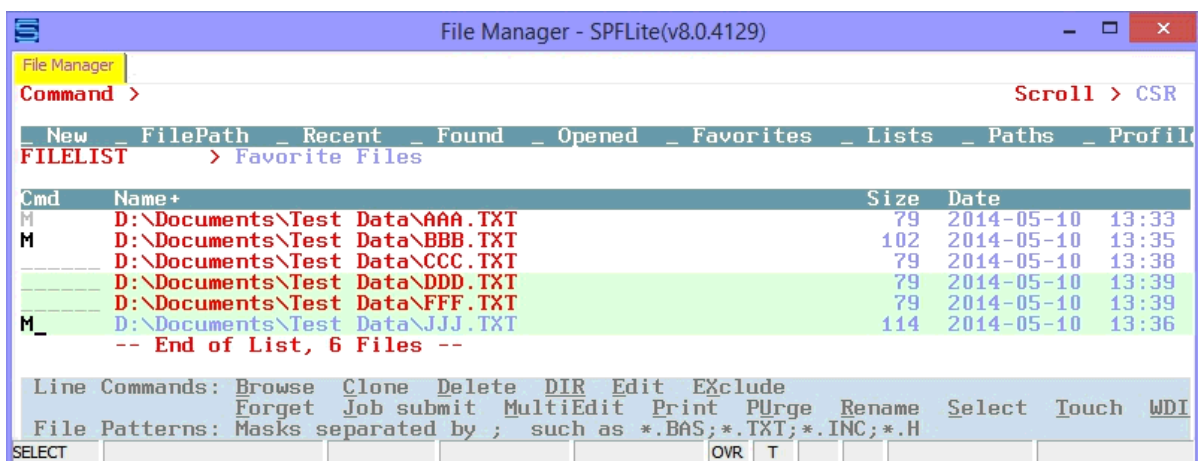
We'll take this one step at a time.

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.



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:

```

(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 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. 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.

It is possible to remove the **=FILE>** marker from the display, if you desire to do so. See the [HIDE](#) primary command.

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 **TJJJ.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:

```

(Multi-Edit) - SPFLite(v8.0.4129)
File Manager (M-Edit)
Command >
Scroll > HALF
CHARS file changed 9 times
***** 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* **=FILE>** 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.

- **M** will allow you to move a file's data to a different position within the MEdit session. The **A/B** destination must be located before another **=FILE>** line or the **** Bottom **** line

Note that a **=FILE>** line itself cannot be typed on, nor can it be changed manually or through primary or line commands.

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.

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.

The command **LOCATE FILE** must not be combined with **ALL** or **MX**.

An alternate method is to assign the [\(MEditList\)](#) primitive to a key. If this key is pressed in a MEdit session, a popup listing all currently loaded files will appear. You may click on any filename, and the screen will scroll directly to that file's position. Here is a sample of the popup.

```

D:\Documents\SPFLite3\Source\ObjParse.inc
D:\Documents\SPFLite3\Source\PCmd.inc
D:\Documents\SPFLite3\Source\LCmd.inc
D:\Documents\SPFLite3\Source\FMPCmd.inc
D:\Documents\SPFLite3\Source\FMLCmd.inc
D:\Documents\SPFLite3\Source\ObjENV.inc
D:\Documents\SPFLite3\Source\Types.inc
D:\Documents\SPFLite3\Source\Mainline.inc
D:\Documents\SPFLite3\Source\InitRoutines.inc
D:\Documents\SPFLite3\Source\BMacro.inc
D:\Documents\SPFLite3\Source\KB.inc
D:\Documents\SPFLite3\Source\ObjSQL.inc
D:\Documents\SPFLite3\Source\ObjSQLM.inc
D:\Documents\SPFLite3\Source\ObjPCmdT.inc
D:\Documents\SPFLite3\Source\ObjKbdT.inc
D:\Documents\SPFLite3\Source\ObjPrimT.inc
D:\Documents\SPFLite3\Source\ObjCmnt.inc
D:\Documents\SPFLite3\Source\DialogEquates.inc
D:\Documents\SPFLite3\Source\ObjLCmdT.inc
D:\Documents\SPFLite3\Source\Resource.inc
D:\Documents\SPFLite3\Source\ObjFMD.inc
D:\Documents\SPFLite3\Source\ObjFCB.inc
D:\Documents\SPFLite3\Source\AsmCode.inc
D:\Documents\SPFLite3\Source\Macros.inc

000024 - Compiler stuff
000025 /
000026 #COMPILE EXE "D:\Documents\SPFLite3\SPFLite3.EXE"
000027 #DIM ALL
000028 #STACK 8388608
000029 #DEBUG DISPLAY ON
000030 #DEBUG ERROR ON
000031 #TOOLS OFF

27 Edit Lines: 68766 Cols 1 to 104 Bnds: MAX INS T DS S+ Profile: INC CAPS OFF ANSI CRLF

```

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 to 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!

NonWindows Text Files

Contents of Article

[Common settings for non-Windows files](#)
[Determining file attributes through experimentation](#)
[EBCDIC files](#)
[Custom Translation Tables](#)
[Using the End-of-Line settings AUTO and AUTONL](#)

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 [DCB 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 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.

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.

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.

SPFLite allows you to process all these types (including files **without** delimiters) by setting the Profile option [DCB EOL](#) to the requirements of your file.

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 [DCB EOL](#) type of **AUTONL**, discussed later in this article.

Character Set Even though the Line delimiter in [DCB 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.

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

UTF BOM Markers

To identify the various types of UTF files, the true data is optionally prefixed with a 2 or 3 byte BOM (Byte Order Marker) This identifies the particular type of UTF coding used.

However some, like UTF8, do not **need** a BOM, and in fact, sometimes the presence of a BOM can cause problems. SPFLite provides a Profile option named BOM (why not?) which can be set to ON or OFF. This controls whether you wish SPFLite to create the BOM when writing the file. See [BOM](#).

Logical Record Length

When the Record Format is set to **F** (for fixed), the length of the fixed record must be specified using the [DCB LRECL](#) command. The *record-length* 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, then try the following:

DCB RECFM U LRECL 0 EOL LF and **SOURCE ANSI**

If it was an IBM mainframe, then try

DCB RECFM V LRECL 0 EOL NONE and

either:

SOURCE EBCDIC

or

DCB RECFM F LRECL nn EOL NONE and
SOURCE EBCDIC

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:

DCB RECFM U LRECL 0 EOL CRLF and
SOURCE UTF8

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.

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 End-of-Line 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 End-of-Line NONE is required.

There is nothing to prevent you from specifying End-of-Line **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 format of the table is straightforward. 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.

The **.SOURCE** format.

Here is an example of the EBCDIC.SOURCE translate table.

```
TT  TITLE='SPFLITE TRANSLATION TABLE'  MODE=RT
TT  GENDATE='2013-11-29 14:36:46'
```

```
**  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	□	œ	09	†	7F	97	8D	8E	0B	0C	0D	□	□
0*																
1*		□				...	08	‡	□		'	•	□	□	□	□
1*																
2*	¤	81	,	f	„	0A		□	^	%	Š	<	œ	05	06	07
2*																
3*	90	`	□	“	”	•	–	□	~	™	š	>	14	15	ž	1A
3*																
4*		A0	â	ä	à	á	ã	å	ç	ñ	ç	.	<	(+	4*
5*	&	é	ê	ë	è	í	î	ï	ì	ß	!	\$	*)	;	¬
5*																
6*	-	/	Â	Ä	À	Á	Ã	Å	Ç	Ñ		,	%	_	>	?
6*																
7*	ø	É	Ê	Ë	È	Í	Î	Ï	Ì	`	:	#	@	'	=	"
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	º	æ	.	Æ	€
9*																
A*	µ	~	s	t	u	v	w	x	y	z	ı	ı	Đ	Ý	Ð	®
A*																
B*	^	£	¥	·	©	§	¶	¼	½	¾	[]	—	…	’	×
B*																
C*	{	A	B	C	D	E	F	G	H	I		ô	ö	ò	ó	õ
C*																
D*	}	J	K	L	M	N	O	P	Q	R	¹	û	ü	ù	ú	ÿ
D*																
E*	\	÷	S	T	U	V	W	X	Y	Z	²	ô	ö	ò	ó	õ
E*																
F*	0	1	2	3	4	5	6	7	8	9	³	û	ü	ù	ú	ÿ
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_																

A_	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	DD	DE	AE
A_																
B_	5E	A3	A5	B7	A9	A7	B6	BC	BD	BE	5B	5D	AF	A8	B4	D7
B_																
C_	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
C_																
D_	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
D_																
E_	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
E_																
F_	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 End-of-Line 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. End-of-Line **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 End-of-Line **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) or lone **CR** (carriage return) 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.
- 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' 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.

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

NOTE and xNOTE Lines

Contents of Article

[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](#)

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**. The last three **xNOTE** lines (WNOTE, XNOTE and YNOTE) are treated as temporary, and will not be saved and restored by [STATE](#) processing.

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. Remember, WNOTE, XNOTE and YNOTE are temporary and are **not** saved by STATE. 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** for blocks) will convert a **=NOTE>** or **xNOTE>** line into a data line.

The **MN** line command (**MNN** 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	Shift cmds	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 can be used to locate lines of type **NOTE**. 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**.

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 note-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** 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 use 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.

Note lines may be 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.

Picture Strings

Special note for $P'f'$ and $P'f'$ or $P'\{f'$ and $P'\{f'$

Picture String Examples

Using Picture strings as the TO operand of a CHANGE command

SPFLite Format Change Strings

Displayable and non-displayable characters

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 \neg can be either the ANSI \neg character (X'AC') or the OEM \neg character (X'AA'). You should select the character that displays properly

with your chosen editor font, but either one will work.

P' #'	Any numeric character, 0-9 Note that the three special superscript characters in ANSI for ¹ , ² and ³ are not considered numeric.
P' - '	Any non-numeric character. which is any character not matching a Picture of P' #'
P' . '	Any non-displayable character; that is, any character not in the Normal list in Global Options
P' @ '	Any alphabetic character, uppercase or lowercase. This includes the non-English letters in the ANSI character set.
P' < '	Any lowercase alphabetic character. This includes the non-English letters in the ANSI character set.
P' > '	Any uppercase alphabetic character. This includes the non-English letters in the ANSI character set.
P' \$ '	Any special character, neither alphabetic nor numeric.
P' % '	Any character not defined in the WORD setting. i.e. a delimiter.
P' & '	Any character defined in the WORD setting. A span of P'&' characters would be a "user-defined word".
P' ! '	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 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 as-is without case conversion.
P' != '	A Picture of P' ! ' may be specified as P' != ' with essentially identical meaning. P' != ' would only be used in cases where a Picture of P' ! ' needed to be followed by a Picture of P' > ' or P' < ' 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 P' != ' notation is like an "escape" that prevents the ! from being confused with P' !> ' or P' !< ' .
P' !> '	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 multiple times in the change-string; if done, the found string is repeated in the result as many times as the ! code is used.
P' » '	The found string is copied and converted to upper case. 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'BB') or the the OEM » character (X'AF'). You should select the character that displays properly with your chosen editor font, but either one will work.

P' !<'

Allowed only in the *change-string*. It represents the **entire** string value actually found (not the string-1 *operand*), **regardless** of its length. It is allowable to use the !< 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.

P' «'

The found string is copied and converted to lower case.

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 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.

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. 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.
- The picture character ! 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 then 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.
- 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.
- Unlike IBM ISPF, SPFLite allows anything in a change Picture which is not one of the “copy codes” to be treated as ordinary data.
- SPFLite introduced 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. This handling of CASE also applies to the normal characters in a Picture string. So, a *search* Picture of P'ABC' is case-*insensitive* if **CASE T** is in effect, case-*sensitive* if **CASE C** is in effect.

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' @@@ '
  _^^^_
P' (===) '
```

Here, you can see where the ^ 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' @@@ '
```

`_^^^_`
`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.

Displayable 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.

Power Typing Mode

Contents of Article

[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 being 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-qualified 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/Overtyping 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.

(Pen/colorname)**(Pen/STD)**

If selected columns have been marked, the appropriate highlight color will be set in those column throughout the entire PT range. Only existing columns will be colored. i.e. a line will NOT be extended with blanks just to be colored.

(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 reset (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 left. 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:

```

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
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 2014-05-11 16:42:13 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+

```

There are a number of ways of starting Power Typing mode on this file. Here is one using a **C/** line command:

```

PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command > PTYPE
Scroll > HALF

***** 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 *****

Edit L 000001 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0016

```

After you press Enter, you will see the Power Typing message and the **PowerType** indicator on the status line:

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 Overtyping 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:

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:

Next, replace the former variable-length text with a fixed-length string that includes the starting

We will highlight columns 6 through 18 to cover the variable-length part of these lines:

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:

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:

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.

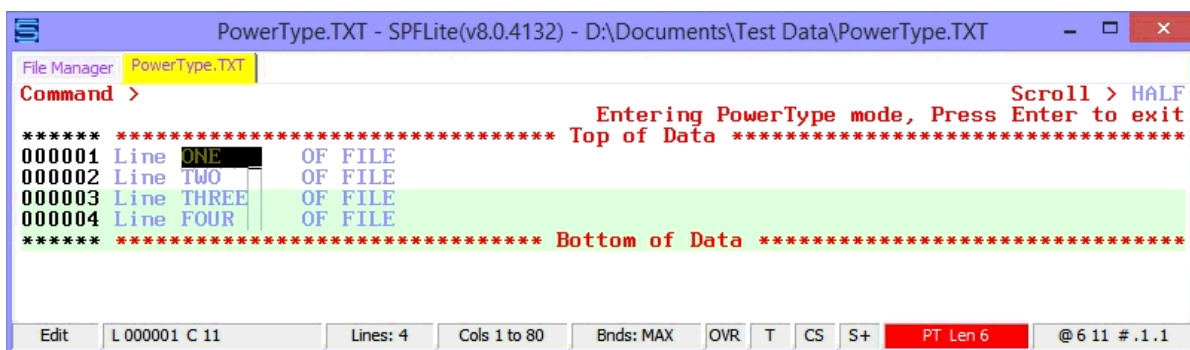
To do this, you will need to map the text-justification functions to keys to make them available. For sake of discussion, let's again assume the following key mappings are in place. As usual, these are not defaults, and you would have to map these keys yourself.

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:

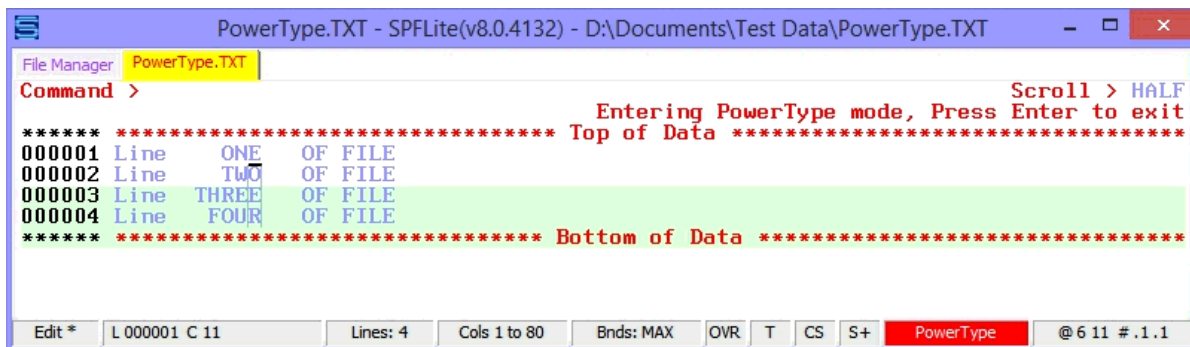
Suppose the task is to put parentheses around each of the words ONE, TWO, THREE and FOUR. Since these words are of varying lengths, how do we do it? First, highlight the columns over the words, so that the highlighting covers the longest word, which in this case is the five letters of the word THREE, 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:



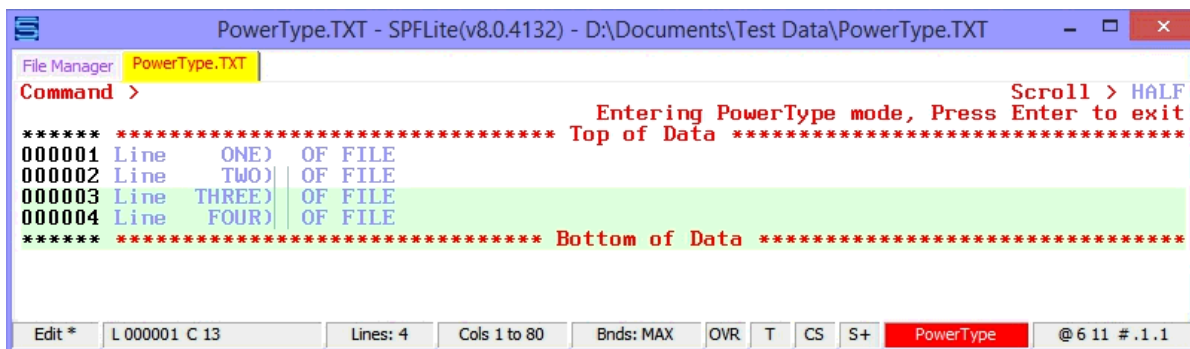
```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command >
Entering PowerType mode, Press Enter to exit
*****
***** Top of Data *****
000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE
*****
***** Bottom of Data *****
Edit L 000001 C 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:



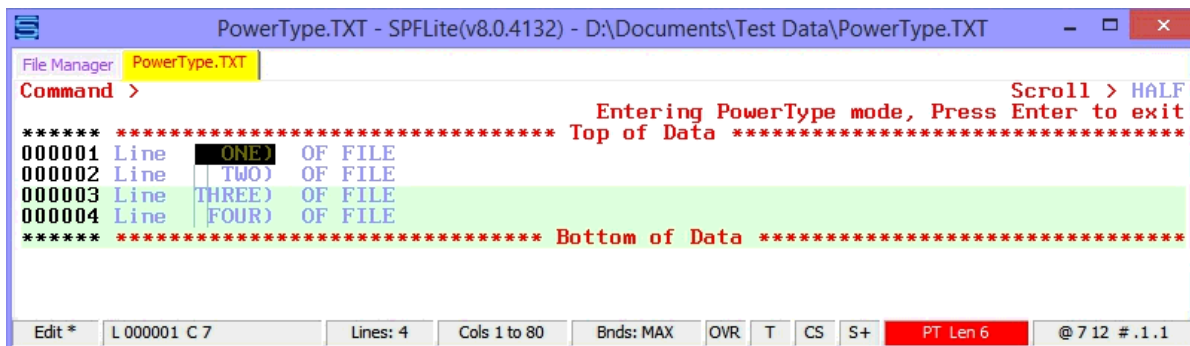
```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command >
Entering PowerType mode, Press Enter to exit
*****
***** Top of Data *****
000001 Line ONE OF FILE
000002 Line TWO OF FILE
000003 Line THREE OF FILE
000004 Line FOUR OF FILE
*****
***** Bottom of Data *****
Edit * L 000001 C 11 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 6 11 #.1.1
```

Move the cursor to column 12 and type a right parenthesis:



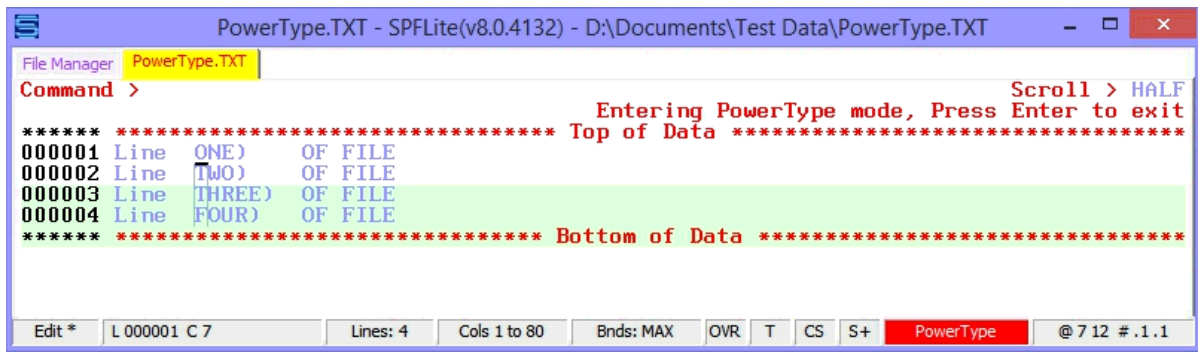
```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command >
Entering PowerType mode, Press Enter to exit
*****
***** Top of Data *****
000001 Line ONE) OF FILE
000002 Line TWO) OF FILE
000003 Line THREE) OF FILE
000004 Line FOUR) OF FILE
*****
***** Bottom of Data *****
Edit * L 000001 C 13 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ PowerType @ 6 11 #.1.1
```

Highlight the columns again as shown here:



```
PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT
File Manager PowerType.TXT
Command >
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+ PT Len 6 @ 7 12 #.1.1
```

Again, noting the keys you mapped above, left-justify the column of text:

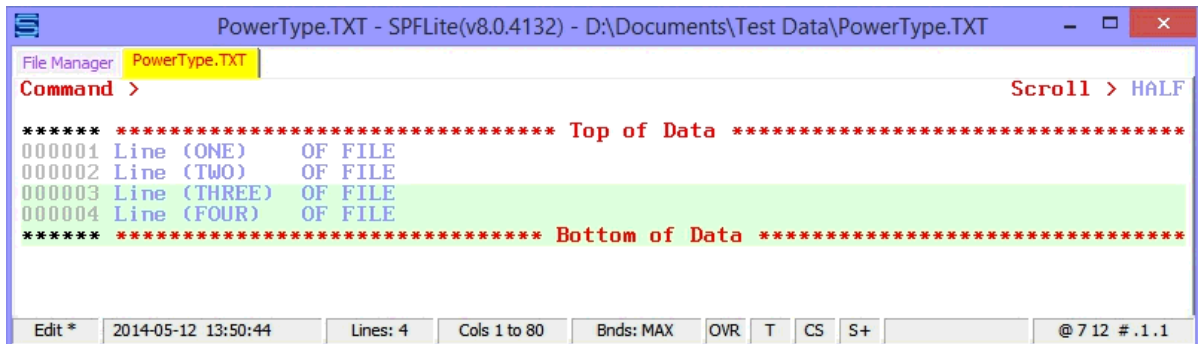


The screenshot shows the SPFLite3 application window titled "PowerType.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\PowerType.TXT". The interface includes a "File Manager" tab, a "Command" prompt, and a "Scroll > HALF" button. The main display area shows the following text:

```
*****  
***** 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 *****
```

The status bar at the bottom displays "Edit *", "L 000001 C 7", "Lines: 4", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", "PowerType", and "@ 7 12 # .1.1".

Finally, move the cursor to column 6, type a left parenthesis, then press Enter to end Power Typing mode. Result:



The screenshot shows the SPFLite3 application window after entering PowerType mode. The main display area shows the following text:

```
*****  
***** 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 status bar at the bottom displays "Edit *", "2014-05-12 13:50:44", "Lines: 4", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", and "@ 7 12 # .1.1".

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 (<code>SPFLiteScrPrt.LOG</code>) 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.

Read Only Files

Contents of Article

[*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 not supported by MEDIT*](#)

[*Actions taken during END or SAVE*](#)

[*Actions taken during Rename or Delete*](#)

[*Actions requiring Windows Explorer or other external intervention*](#)

Introduction

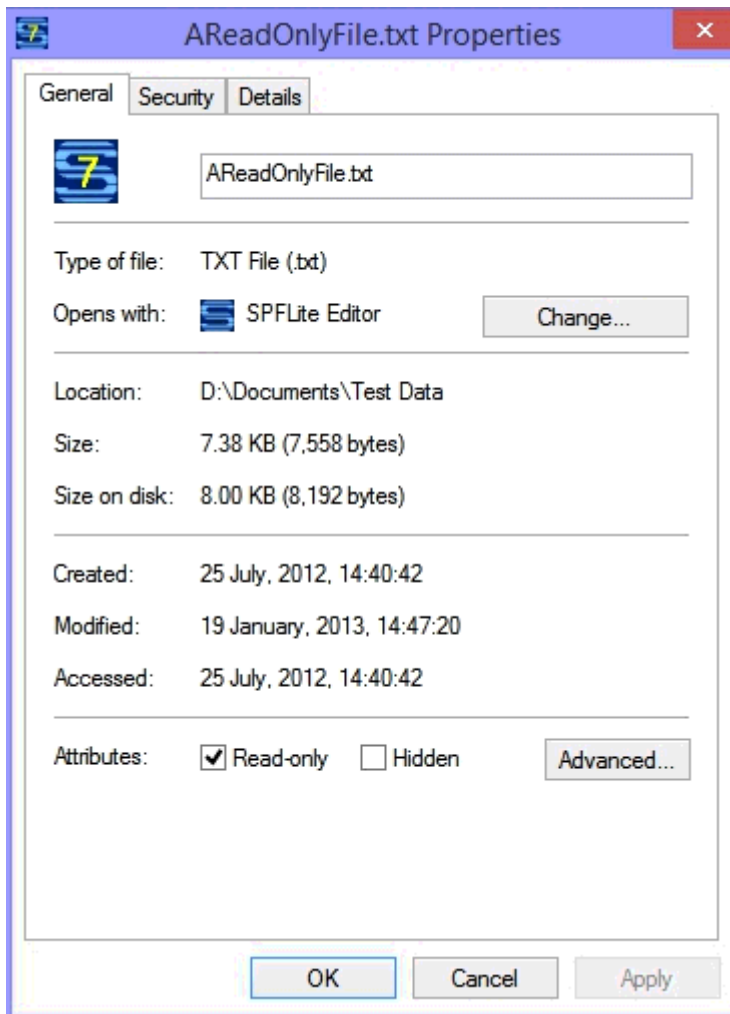
SPFLite will respect the Read Only attribute of a file. Because this action takes place automatically, there is not a lot for you to do, except to understand the process SPFLite uses to handle Read Only files.

Determining that a file has the Read Only attribute

From the Windows Explorer

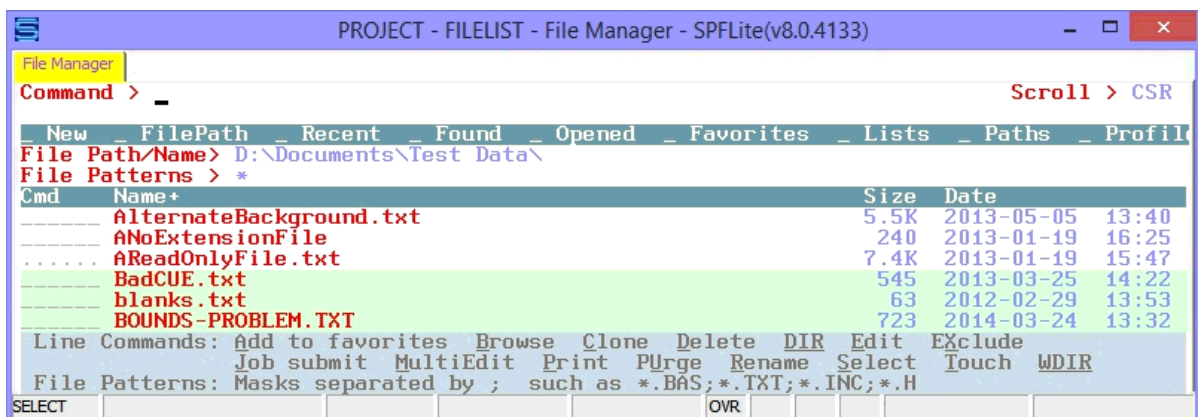
When Explorer displays a list of files, you can see the Read Only attribute (among other things) if you add the **Attributes** column heading to the display. To do that, right-click on the Explorer heading line on some empty area, and a box of heading types will appear; click on **Attributes**. This 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:



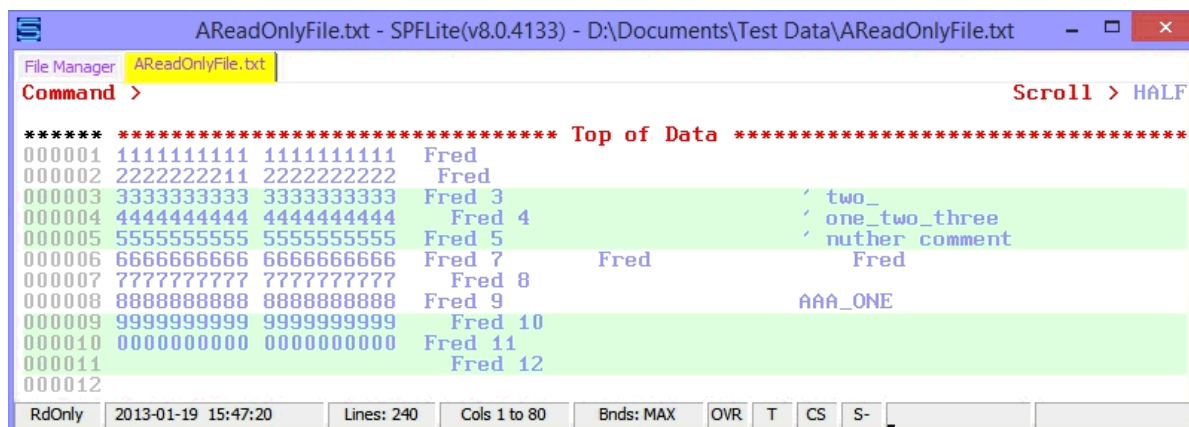
Accessing Read Only files from the SPFLite File Manager

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:



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.



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 not supported by 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** 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.

Regular Expression

Contents of Article

[Performance Considerations](#)

[Examples of Using RegEx](#)

[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)
\odd character with octal code odd
\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 **\odd** 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
Li	Lower case letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter
Lu	Upper case letter
L&	Li, Lu, or Lt
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
NI	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	Univerally-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.

Examples of Using RegEx

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-z0-9._/+-]+) (@[a-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.,]+"

Scrolling the Text

Contents of Article

[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.

Shifting Data

Contents of Article

[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 quotes (single or double) 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.
- IBM ISPF documentation states that both single 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.

Because data shifts may compress or move spaces around, SPFLite will attempt to logically assign colors based on the attributes of the surrounding 'words'. 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.

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 **]n** 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 $4 \times 3 = 12$. 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 toggle 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 hilights as **INS** instead.

When the editor is in Data Shift Insert Mode, pressing [\(DataInsert\)](#) again will toggle Data Shift Insert mode off, and return the previous state of the Insert key.

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.

SPFTest Program

The use of Regular Expression 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.

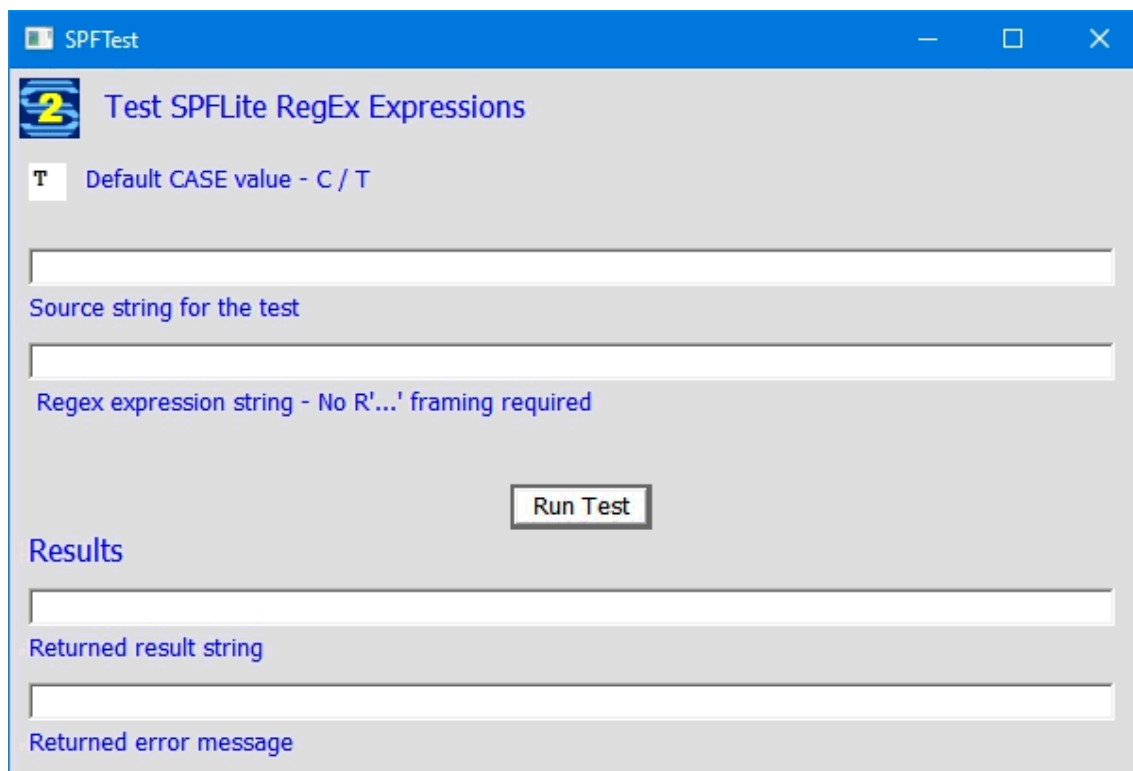
The SPFTest program was design to provide you with a 'scratch pad' for experimenting with Regular Exprssions, and immediately see results, without having to set up test data in an SPFLite edit session. Since these tests don't involve actual data, you can't harm anything by running these tests, and you 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 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.

Regular Expression Strings



SPFTest

Test SPFLite RegEx Expressions

T Default CASE value - C / T

Source string for the test

Regex expression string - No R'...' framing required

Run Test

Results

Returned result string

Returned error message

This is the testing interface for testing Regular Expression strings. The top half of the

screen contains three input fields.

Default Case	<p>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 <code>for</code> for case sensitive or <code>T</code> 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.</p> <p>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.</p>
Source string	<p>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.</p>
Regex Expression	<p>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.</p>

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. The lower box will contain information about the located match string, or it will contain any error messages if the test is unsuccessful.

The Regular Expression syntax now accepted is that of the well-known, open-source PCRE regular expression engine.

For a full explanation of Regular Expressions, see [Specifying a Regular Expression](#).

SPLIT and JOIN Commands

Contents of Article

[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.

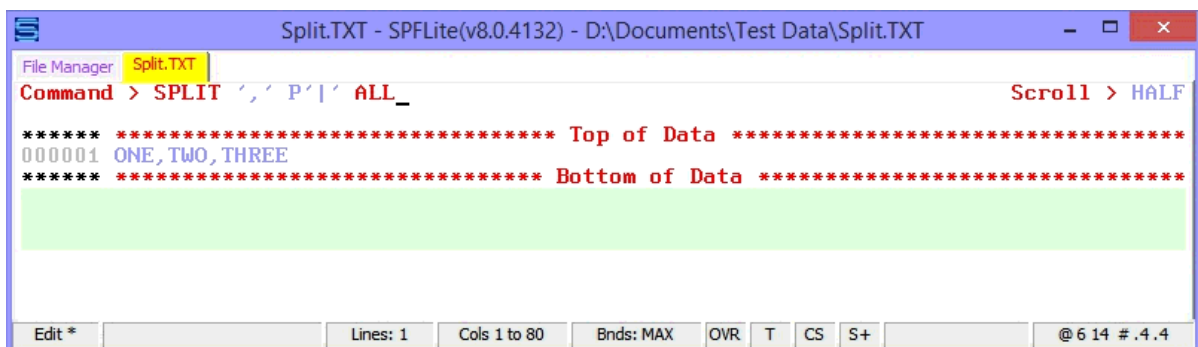
R 'expression\$'

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



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.

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 0003

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.

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 3: Split comma-separated values, retaining commas after split

The screenshot shows the SPFLite3 application window titled 'Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT'. The 'File Manager' tab is active, showing 'Split.TXT'. The 'Command' field contains the command: `> SPLIT ', ' P'|, ' ALL_`. The status bar at the bottom indicates 'Lines: 1', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+'. The main display area shows the following text:

```

***** Top of Data *****
000001 ONE,TWO,THREE
***** Bottom of Data *****

```

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.

The screenshot shows the SPFLite3 application window titled 'Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT'. The 'File Manager' tab is active, showing 'Split.TXT'. The 'Command' field is empty. The status bar at the bottom indicates 'Lines: 3', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', 'Line Len 0004'. The main display area shows the following text:

```

***** Top of Data *****
000001 ONE
000002 , TWO
000003 , THREE
***** Bottom of Data *****

```

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.

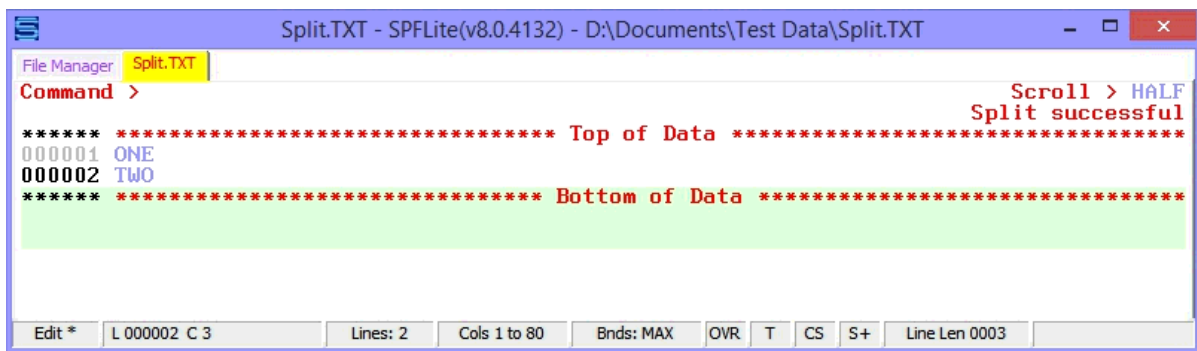
The screenshot shows the SPFLite3 application window titled 'Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT'. The 'File Manager' tab is active, showing 'Split.TXT'. The 'Command' field contains the command: `> SPLIT ONETWO p'ONE|TWO'`. The status bar at the bottom indicates 'Lines: 1', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+'. The main display area shows the following text:

```

***** Top of Data *****
000001 ONETWO
***** Bottom of Data *****

```

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.



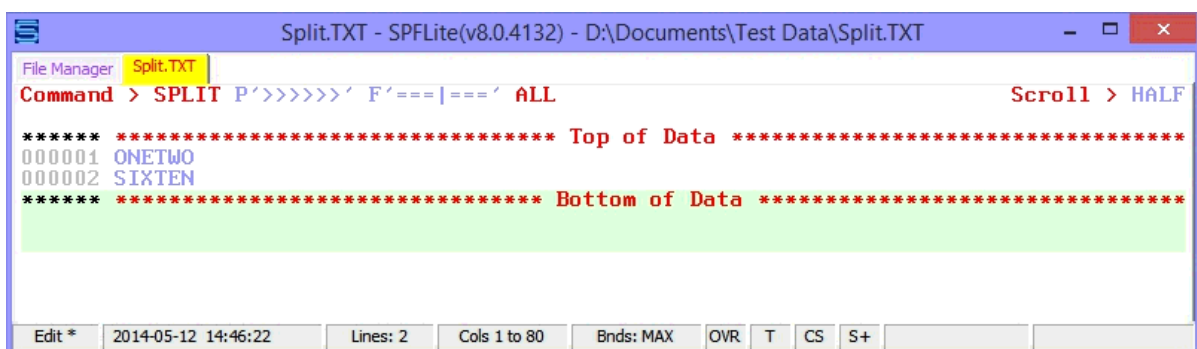
Example 5: Split using Picture and Format strings

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.



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.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > **SPLIT - F ALL** 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**.

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|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.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > **SPLIT ; F ALL** 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**.

```

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=1;two=2;TWENTY=20;three=3;
***** Bottom of Data *****

Edit * 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+ @ 33 33 #.1.1

```

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.

```

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT
File Manager Split.TXT
Command > Split performed 3 times
***** Top of Data *****
000001 one=1;
000002 two=2;TWENTY=20;
000003 three=3;
000004
***** Bottom of Data *****

Edit * L 000002 C 1 Lines: 4 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0016

```

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_ Scroll > HALF
***** Top of Data *****
000001 one=ABCtwo=Abctthree=abc
***** Bottom of Data *****

Edit * 2014-05-12 14:46:22 Lines: 1 Cols 1 to 80 Bnds: MAX OVR T CS S+

```

Result:

The screenshot shows the SPFLite3 application window titled "Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT". The "File Manager" tab is active, showing the "Split.TXT" file. The "Command" field contains ">". The status bar at the bottom indicates "Edit *", "L 000002 C 1", "Lines: 4", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", and "Line Len 0007".

```

*****
000001 one=ABC
000002 two=Abc
000003 three=abc
000004
*****
  
```

Annotations in the image include "Scroll > HALF" and "Split performed 3 times" in the top right, and "Top of Data" and "Bottom of Data" in red text within the data area.

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.

The screenshot shows the SPFLite3 application window titled "Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT". The "File Manager" tab is active, showing the "Split.TXT" file. The "Command" field contains "> SPLIT R'^[A-Z]+' F'!|' ALL_". The status bar at the bottom indicates "Edit", "2014-05-12 15:11:14", "Lines: 6", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", and "Line Len 0007".

```

*****
000001 one111
000002 Two+222
000003 three3333
000004 444FOUR444
000005 eleven 11
000006 seventeen**1717**
*****
  
```

Annotations in the image include "Scroll > HALF" and "Top of Data" and "Bottom of Data" in red text within the data area.

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.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > Scroll > HALF

Split performed 5 times

***** 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.

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > SPLIT P'[ABC ' P'DEF|' ALL_ Scroll > HALF

***** 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+

Result:

Split.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Split.TXT

File Manager Split.TXT

Command > Scroll > HALF

Split performed 3 times

***** 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_

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 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 >

Scroll > HALF

Join successful

***** 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

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_

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 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.

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 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
```

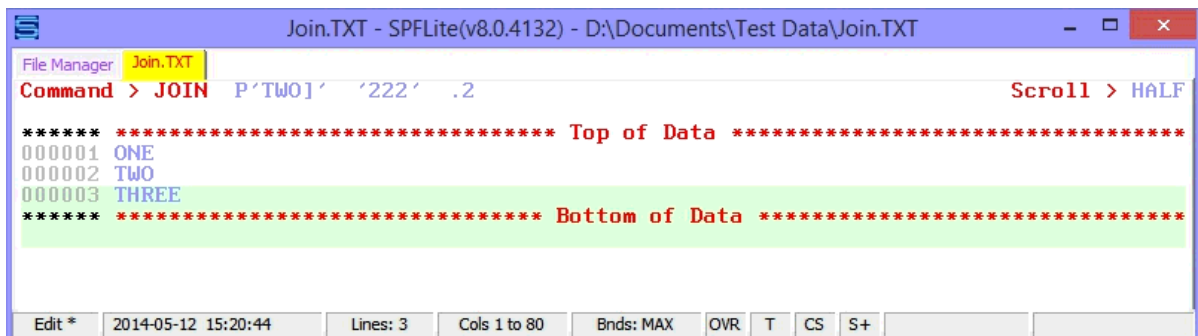
Try it yourself, and convince yourself that this works. We can call this a "simple join".

Now, internally this is what is really happening:

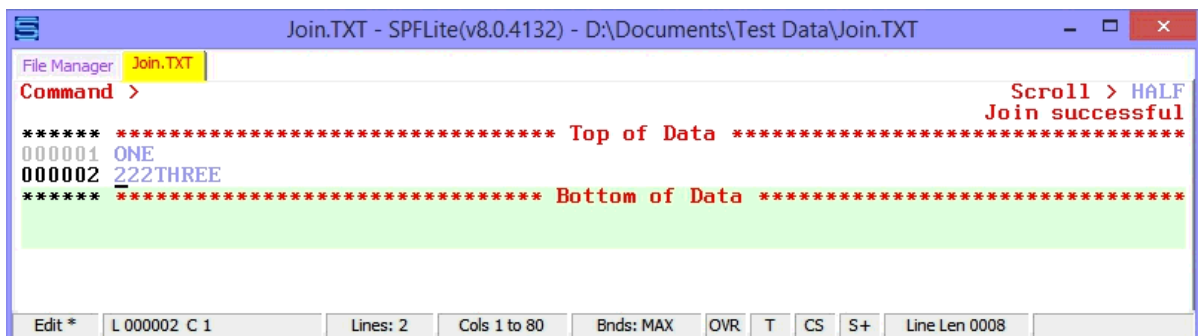
- A line with "TWO" at the end is found (the **]** right bracket means "at the end")
- The string "TWO" is deleted from the end of line 2
- The contents of line 2 (with the ending "TWO" removed) **plus** the string "TWO" **plus** the contents of line 3 are concatenated together to make a new line 2.

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.



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.



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 it, 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.

```

Join.TXT - SPFLite(v8.0.4132) - D:\Documents\Test Data\Join.TXT
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 > Join performed 2 times Scroll > HALF
***** 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.


```

File Manager join.txt
Command > JOIN P';] ' ; ' ALL Scroll > HALF
***** 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+

```

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.

```

File Manager join.txt
Command > Scroll > HALF
Join performed 3 times
***** 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' =] ' ' P' = ' ALL** 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".**

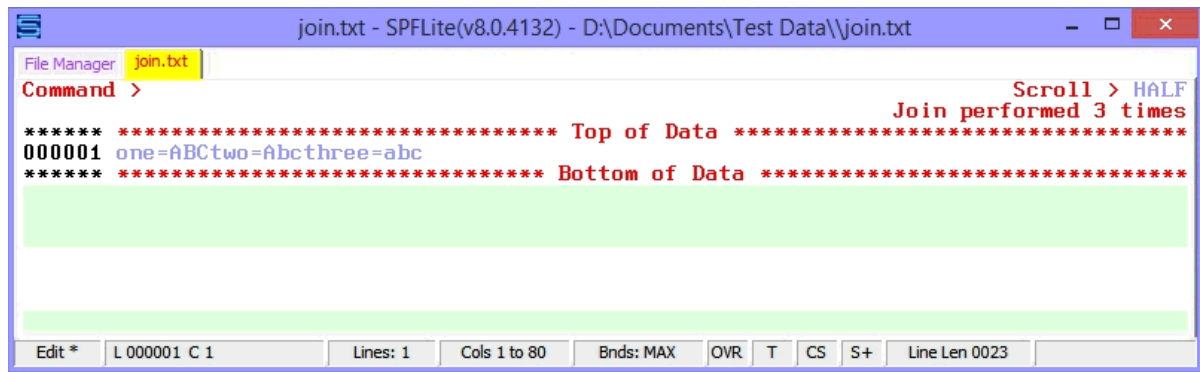
```

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.



The screenshot shows the SPFLite3 application window titled "join.txt - SPFLite(v8.0.4132) - D:\Documents\Test Data\join.txt". The window contains a text editor with a command prompt interface. The command prompt shows the command "Command >" and the output "Join performed 3 times". The text editor displays the following content:

```
*****  
***** Top of Data *****  
000001 one=ABCTwo=Abctthree=abc  
*****  
***** Bottom of Data *****  
*****
```

The status bar at the bottom of the window shows the following information:

Edit *	L 000001 C 1	Lines: 1	Cols 1 to 80	Bnds: MAX	OVR	T	CS	S+	Line Len 0023
--------	--------------	----------	--------------	-----------	-----	---	----	----	---------------

STATE Saving

Contents of Article

[STATE and the File Type](#)

[STATE Integrity](#)

[STATE Command Options](#)

[Displaying the STATE setting in the Profile](#)

[Where is the STATE saved ?](#)

[When is STATE data 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 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 session 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, the line count when the STATE data is created is matched against the line count when the file is loaded. If the validation 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. 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.

SPFLite Backup / Restore Support

The built in Backup/Restore support in SPFLite will create backups which combine the Date and STATE files together to maintain integrity between the files. See "[Working with BACKUP & RESTORE](#)" for details.

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 allows you to 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.
STATE SAVE	This command will immediately create / update the STATE information for the current file. It will be performed regardless of the current setting of the STATE flag (ON, OFF, etc.)

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:

```

***** 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>
=MARK>
=MARK>
=TABS> * * +
=COLS> _ _ _ _ _ 1 _ _ _ _ 2 _ _ _ _ 3 _ _ _ _ 4 _ _ _ _ 5 _ _ _ _ 6 _ _ _ _ 7 _ _ _ _
=BND> <
000001 Line one
-----
000004 Line four
-----
***** Bottom of Data *****

```

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 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 removed from the STATE folder. This will prevent the accumulation of obsolete STATE files which are no longer of any use.

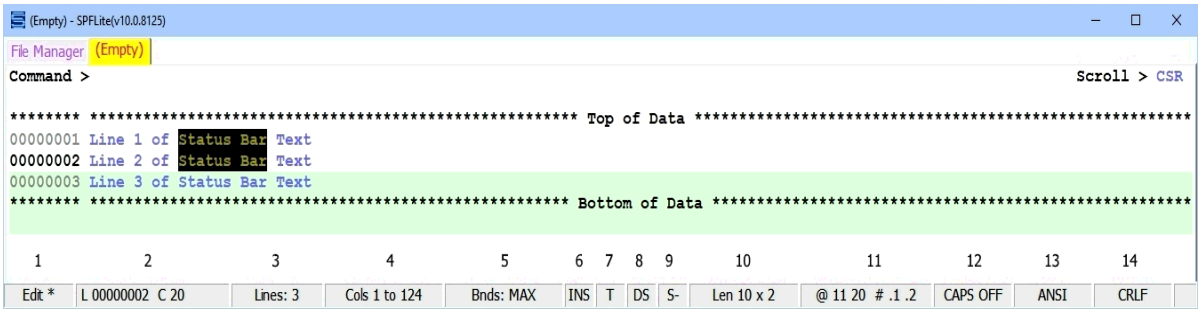
When is STATE data saved?

The STATE data is saved a) whenever the actual data is Saved and b) when the session is closed with an END command and the data is in un-modified state. This saving at END time is done for all session types (Edit, Browse and View)

If the session is closed with a CANCEL, no STATE data is saved.

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**, **EFT Edit** 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. You may also see **EFT Edit** or **Set Edit** while performing these special edit types.. 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***

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 = Overtyping mode. If the current setting differs from your chosen default INS/OVR setting (in OPTIONS -> Keyboard) the value will be in reversed colors to remind you things are not 'normal' . 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 the vertical scroll bar (if used).

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 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.

If you set **CAPS** mode to **AUTO**, the caps mode is “conformant”. The file is examined, and set to Caps ON or OFF based on whether the file data is mixed case or not. This **ON/OFF** status 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.

Box 13 The current character set being used. This is controlled by the [SOURCE](#)
(SOURCE) Profile setting. The default is ANSI. Other possible values are EBCDIC and various Unicode settings.

Box 14 The current [EOL](#) setting. This shows how the End Of Line value has been
(EOL) 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 **DCB** 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.

SUBMIT Command

Contents of Article

[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 CMD 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.

SPFLite Global Options (DEFAULT)

General | FM | **Submit** | Screen | KBD | Status | Schemes | Hilights | Config

Prototype command line to be used by SUBMIT


Working Directory to be used by SUBMIT

CMD.EXE flags used by SPFLite CMD command

CMD.EXE flags used by SPFLite RUN command

Trigger key for the SUBMIT Include statement

Column number where above Include Key must appear

 SPFLite.CFG file Folder:

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 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.

Tab Pages

Contents of Article

[Data isolation](#)

[Switching between tabs](#)

[Opening a new tab](#)

[Closing a tab](#)

[Indicating file-modified status in the tab header](#)

[Example Appearance](#)

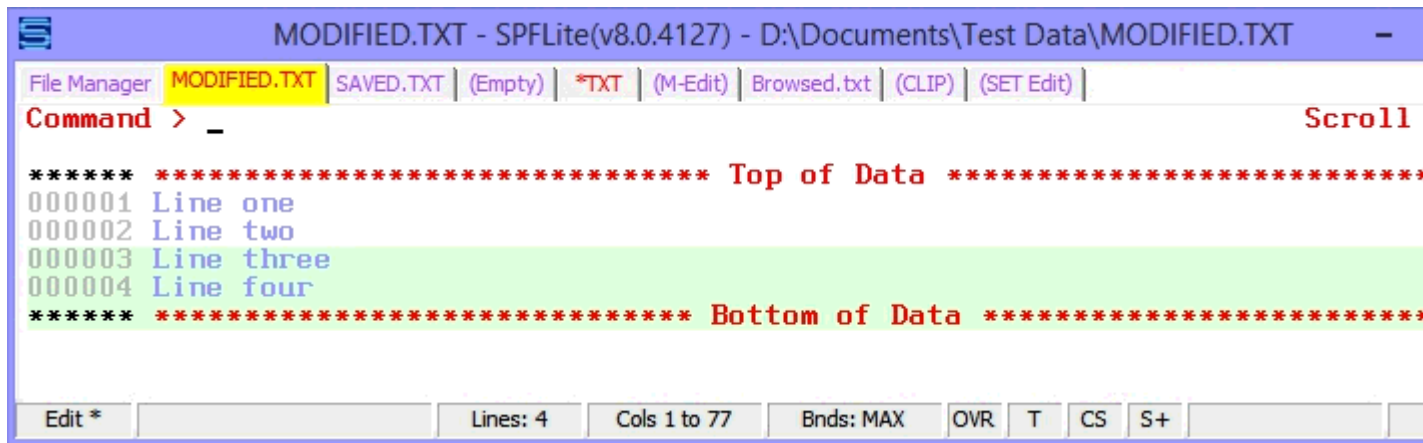
Introduction

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.
- **New 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.
- **EFT Edit tab.** Contains the values for the current EFT rule set. At most one EFT 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 a 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**, **V** 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.

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.

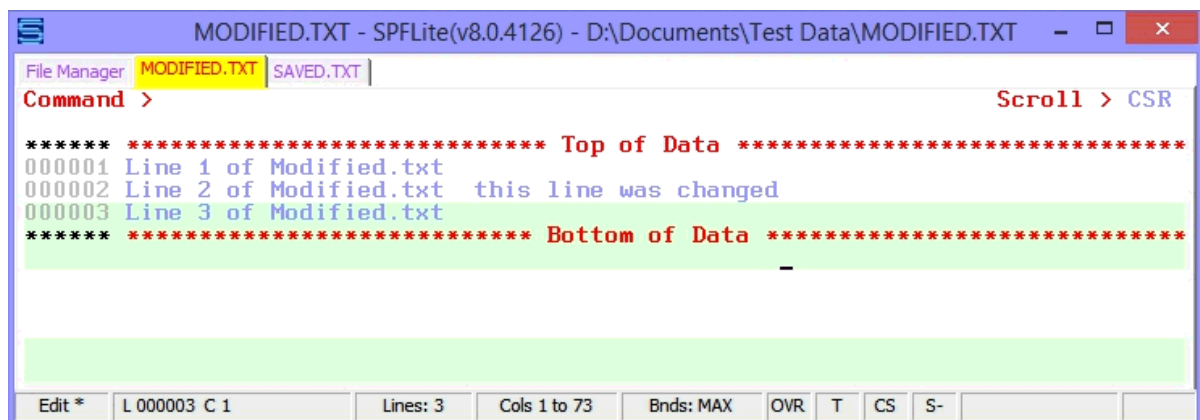
The colors to be used are under your control, and are specified in the [Options - Screen](#) dialog.

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:



Example Appearance

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**.



Tabs and Column Markers

Contents of Article

[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 data may be for language syntax indenting, for standard positioning of comments, or for any other reason where standard positions for data is desirable.

Tabs

Tabs are markers indicating the horizontal positions to which the TAB key will successively position the cursor. 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 the operation of the TAB key; the two are independent of each other. They are intended to provide visual guidance only. For example, a column is typically aligned at a column marker because it's usually convenient to combine them that way. Most files having many tab stops will only have a few column marker lines defined, but they probably would place those 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
=TAGS>      *      *
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. Each time this file type is edited, the Tabs and Mark lines will be re-established when the file is loaded into the editor.

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 line, 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 position. For example, if you wanted tab stops in columns 1 and 10, and every 5 columns starting in column 16, entering 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 the *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 will code in column 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 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 the screen to be sure they were visible in this Help document. In practice, it is helpful to make these very faint so you can barely see them; that way they can act as a guide but not be a distraction. The exact color for the MARK lines is in the [Screen tab of Global Options](#) under Column Marker Line. As can be seen, the Mark lines appear on actual data lines, not on special lines such as the =TABS> and =MARK> lines themselves.

```

Mark_Tabs.txt - SPFLite(v8.0.4126) - D:\Documents\Test Data\Mark_Tabs.txt
File Manager Mark_Tabs.txt
Command > _
***** ***** Top of Data *****
000001 Data |Text Here |
=TABS> * * +
000002 More |Data Text Here |
=MARK> * < ><
000003 | Very Much More
***** ***** Bottom of Data *****
Edit * Lines: 3 Cols 1 to 77 Bnds: MAX OVR T CS S-

```

Tab Bounds Mode

Introduction

SPFLite is often used to edit data which is in a tabular format, or columns. If a [TABS](#) line is established for the file type, it can assist in moving the cursor between the various columns. But normal TABS processing only looks after moving the **cursor**. When editing the data it is all too easy when inserting or deleting characters to shift and mis-align data in columns to the right.

Tab Bounds Mode

Tab Bounds Mode is designed to assist editing of columnar data by using the tabs defined in the [TABS](#) line to define the columns and allow editing of data in a column without disturbing data in columns to the right.

It is entirely optional, and if Tab Bounds Mode is never turned on, there is no difference to normal editing activities.

To turn it on, enter the [\(TabBNDS\)](#) keyboard primitive, which you must of course, assign to whatever key combination you prefer. When activated, it will be indicated in the StatusBar INS/OVR box. This box normally contains OVR, INS or DIN, When Tab Bounds Mode is on it will show TAB OVR, TAB INS or TAB DIN. The (TabBNDS) key is a toggle, repeated presses will alternate between ON and OFF.

TabBNDS mode will also be automatically reset by the following KB functions: (SetINS) (SetOVR) (SetDIN) and (Clearinsert)

When active, the following changes are made to normal KB functions.

- | | |
|--|---|
| (Tab) (BackTab) | No change to either, they will perform as usual. |
| (Delete) (DataDelete) (DataBackspace) | These functions will perform their normal functions within the boundaries of the current column in which the cursor is located. |
| (EraseEOL) | The erase will be to the end of the current column in which the cursor is located. |
| Typing in TABOVR mode | In TABOVR mode there is no change to normal typing, the Tab boundaries have no effect. |
| Typing in TABINS or TABDIN mode | Characters typed are entered at the cursor location, and following characters in the column are pushed right as long as there are trailing blanks in the column . If the last position in a column is non-blank, you cannot enter any more characters in these modes, as this would impact the following column(s). If you attempt to do this, it will be rejected with a BEEP signal (either audible or visible, or both depending on your options) See (TabRelease) for more info. |

Note: If **ALL** following columns are blank, the column boundaries are ignored.

- (TabRelease)** If you have entered more characters in insert mode and received the BEEP, warning you of a column overflow, you can correct it by removing characters from the column (if possible), or enter a (TabRelease) key, which will suspend Tab Bounds mode for the remainder of this line. Once the cursor moves away from this line normal Tab Bounds Mode processing resumes.
- (TabShift)** (TabShift) - which you must of course assign to a key, will shift the data from the cursor location to the next Tab boundary. If possible, because some columns to the right are blank, other data to the right will be kept in place wherever possible.

Created with the Personal Edition of HelpNDoc: [Experience the power of a responsive website for your documentation](#)

Tracking Screen Positions

Introduction

Most editing tasks, particularly those involving editing source statements, end up jumping around the file. Yes, a label can be assigned to 'where you are' - e.g. .HERE so that later you can do a LOCATE .HERE to go back to where you were and continue editing. Or you could just remember the current line number to go back to.

This 'jump' to another location may simply be to review code in another location, or perhaps to copy a code fragment for re-use. But interrupting your thought patterns to remember a line number, or set a label is disruptive.

What if this setting of a return point was painless and automatic? Well, with SPFLite tracking support, it now is.

Many of the built-in commands will now automatically create a 'Track Point' before the command is processed, and another 'Track Point' when the command completes. This includes commands such as FIND, LOCATE, CHANGE, RFIND etc.

How to "Go Back"

Returning to your previous location is now trivial. A single key. There are three new KB primitives - (TRACK) (TRACKF) and (TRACKC).

(TRACK)

This will return you to the position in Edit of the most recent 'Track Point'. The screen will be positioned to the same Top-Of_Screen location, and the cursor placed where it was when the command was initiated.

Note: If you press (TRACK) and the cursor is still sitting exactly where the most recent 'Track Point' would take you (i.e. nowhere) then the next previous Track Point' will be used.

(TRACKF)

This companion operation is for those cases where you simply hit (TRACK) too many times. It goes Forward again in location.

(TRACKC)

This will create a new Track Point at the current screen location. Intended for use mainly in KB macros where the other commands may not cause a Track Point to be created.

What about Macros, or Other Commands

The built-in list of commands that automatically "track" can be extended to include whatever other commands and/or macros you wish. This is done using the SPFLite SET facility. To add a macro or command to this list simply:

Issue the command `SET TRACK.cmdname = Y`

at the SPFLite command line.

What about SPF_CMDs issued by Macros

The tracking support will ignore tracking during macro execution. e.g. if the macro issues FIND or LOCATE commands, those internally issued commands will **NOT** create track entries. A Track entry is created before and after the macro is run (assuming a TRACK.macroname = Y has been created) but no new Track entries will be created **during** macro execution.

How far back can I go?

Currently the backtrack limit is 200 positions. This should be more than sufficient for any normal editing requirement.

Can Macros access the internal TRACK support?

Yes, three macro functions are available.

Get_Trk_Pos\$(index-number)

The index-number indicates which Track position is desired, where: (1) indicates the latest, (2) indicates the prior entry, (3) the next previous, etc. If the index number is not specified, the default is (1).

The answer returned is a simple comma delimited String of the format

TOS-Line,Csr_LPTr,CSR_Col

Where:

TOS-Line	The Lptr of the Top-of-Screen line in character format.
Csr_LPTr	The Lptr of the line containing the cursor in character format. If this value is zero, it indicates the cursor was on the command line.
Csr_Col	The location of the cursor within the text line in character format. If zero it indicates the cursor was in the Line-number area.

The Return-code should be checked before using the return value, since errors are possible. Currently the two error conditions are:

- The index-number references a Stack entry which has never been used.
- The Stack entry has been used, but now references a Top-of-Screen line which no longer exists (i.e. it has been deleted)

Get_Trk_TrkID(line-pointer)

Every line in an Edit session has a permanently assigned Track ID. The ID is assigned once, and is never changed or removed. Obtaining this ID allows a macro to obtain a guaranteed pointer to a specific data line.

The operand is a normal *line-pointer* (NOT a Line Number)

The answer returned is the assigned Track ID (a simple numeric value)

The Return-code should be checked before using the return value, since an invalid *line-pointer* will simply return a zero. This could occur if the *line-pointer* references a line which has been deleted.

Get_Trk_LPtr(track-ID)

This function is the reverse of Get_Trk_TrkID, it accepts a *track-ID* as an operand, and returns the *Lptr* of the line.

The operand is a *track-ID* obtained by a prior Get_Trk_TrkID call.

The answer returned is a the Lptr for the requested ID.

The Return-code should be checked before using the return value, since the *track-ID* could be for a line which has been deleted.

User lines

Contents of Article

[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 of USER Lines](#)

Introduction

Conceptually, all data lines exist in one of two states: either they are "**U lines**" or they are "**NU lines**", where **U lines** are simply a set of one or more lines of special interest to the user at some given time, and **NU lines** are everything else.

Essentially, the U/NU state of a line is like a **bit**: It's **on** if it's a U line, and **off** if it's an NU line.

A file will ordinarily consist entirely of **NU lines**, which is the default state of a file when no lines have been marked as U lines. You will see the expression "**ordinary NU 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/NU states, there is no difference between U and NU 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/NU 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 NU lines once the file is closed and reopened.

Note: Because the U/NU 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.

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/NU 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/NU state back to **NU**, 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**

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 NU 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, 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.

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 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
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</i> MX 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
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.

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/VV** line commands, and make no other changes to this line, the : colon in the gap column will reappear.

Examples of USER Lines

Let's start with a simple test file. Notice there is a 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 > Scroll > HALF

```

***** 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+

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.)

User_Lines.TXT - SPFLite(v8.0.4129) - D:\Documents\Test Data\User_Lines.TXT

File Manager User_Lines.TXT

Command > UU P'(<<<<' ALL_ Scroll > HALF

```

***** 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+

This will mark the desired lines, and then report on the number of lines affected:

User_Lines.TXT - SPFLite(v8.0.4129) - D:\Documents\Test Data\User_Lines.TXT

File Manager User_Lines.TXT

Command > Scroll > HALF

```

***** Top of Data *****
000001 | (one)
000002 | (two)
000003 | (six)
000004
000005 (ONE)
000006 (SIX)
000007 (TWO)
***** Bottom of Data *****

```

Edit L 000001 C 5 Lines: 7 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0009

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 screenshot shows the SPFLite3 interface with the file 'User_Lines.TXT' open. The Command window displays the command: `Command > F ')' ' ALL U_`. The status bar at the bottom shows 'Edit', '2014-05-10 13:11:07', 'Lines: 7', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', and 'Line Len 0009'. The main display area shows the contents of the file, with lines 1-3 highlighted in green. The text in the display area is as follows:

```

***** Top of Data *****
000001 | (one)
000002 | (two)
000003 | (six)
000004 |
000005 | (ONE)
000006 | (SIX)
000007 | (TWO)
***** Bottom of Data *****

```

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 screenshot shows the SPFLite3 interface with the file 'User_Lines.TXT' open. The Command window displays the command: `Command >`. The status bar at the bottom shows 'Edit', 'L 000001 C 9', 'Lines: 7', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', and 'Line Len 0009'. The main display area shows the contents of the file, with lines 1-3 highlighted in green. The text in the display area is as follows:

```

***** Top of Data *****
000001 | (one)
000002 | (two)
000003 | (six)
000004 |
000005 | (ONE)
000006 | (SIX)
000007 | (TWO)
***** Bottom of Data *****

```

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:

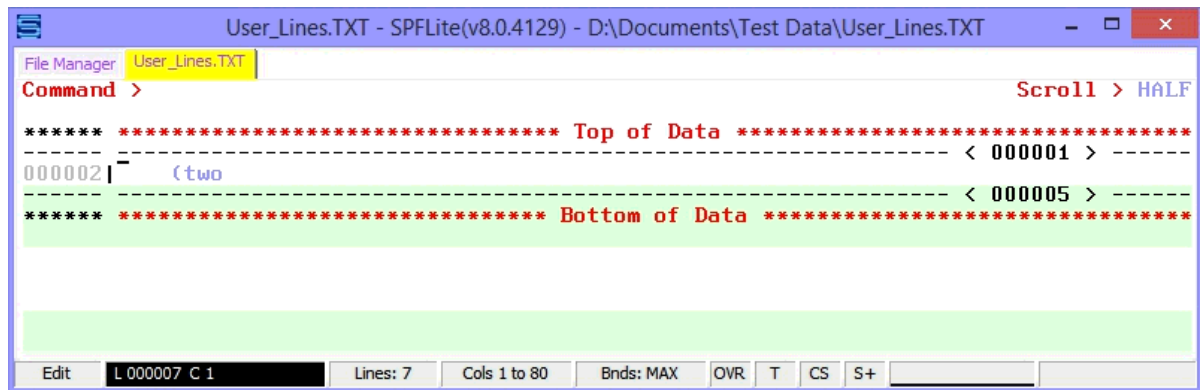
The screenshot shows the SPFLite3 interface with the file 'User_Lines.TXT' open. The Command window displays the command: `Command > X ALL; NFIND ')' ' ALL U_`. The status bar at the bottom shows 'Edit', '2014-05-10 13:11:07', 'Lines: 7', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', and 'Line Len 0009'. The main display area shows the contents of the file, with lines 1-3 highlighted in green. The text in the display area is as follows:

```

***** Top of Data *****
000001 | (one)
000002 | (two)
000003 | (six)
000004 |
000005 | (ONE)
000006 | (SIX)
000007 | (TWO)
***** Bottom of Data *****

```

Sure enough, once you do that, the only line left displayed is the one with the missing right parenthesis on line 2:



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](#).

Virtual Highlighting Pens

Contents of Article

[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 (Hilight) a section of text in any of 15 different colors.

The name **STD** is used to 'clear' a color hilight. Setting a hilight consists of marking the desired area and pressing a key to which a (Pen/*colorname*) 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 highlighted **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 it's current color) be hilighted in tthe 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 highlight colors can be tailored in the Options -> Highlights 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
file

-- where 4 means "red" in an **.AUTO**

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 Highlight actions.

Avoid confusing yourself with clashing colors

The exact colors displayed by the highlighting pens can be adjusted in the Options -> Highlights 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 (afiletype.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:

The screenshot shows the SPFLite3 interface with the file 'HighLight_Pen.TXT' open. The text content is as follows:

```

***** ***** 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 *****
  
```

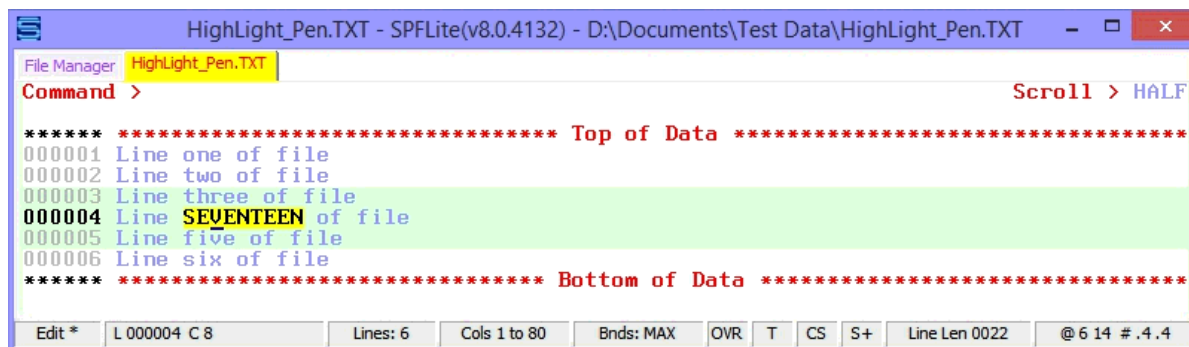
The status bar at the bottom indicates: Edit | 2014-05-12 14:17:24 | Lines: 6 | Cols 1 to 80 | Bnds: MAX | OVR | T | CS | S+.

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:

The screenshot shows the same file as before, but now the word 'SEVENTEEN' on line 4 is highlighted. The status bar at the bottom indicates: 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:



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.

A handy key mapping

If you set a key map (say Alt-Y) to **(CMarkRight) (Pen/Yellow) (Right)** it allows you to very simply mark a string at the current cursor location. It uses the **(CMarkRight)** primitive, which is a *conditional* mark. i.e. if an area is currently marked, it does nothing, otherwise it performs a normal **(MarkRight)** function. So what does this buy you?

If nothing is currently marked, just press Alt-Y and you will hilight the current character and be positioned so that you can repeat the Alt-Y key and mark text 1 character at a time.

If you already have an area marked, it simply hilites the whole marked area.

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.

Word and Delimiter Characters

Contents of Article

[Setting the WORD characters](#)
[Setting the Invalid Character string for P'.'](#)
[Character String Syntax](#)

Introduction

Several SPFLite functions require a list of user provided characters to make decisions. For example the [WORD](#) support uses a list to determine whether a character is part of a "word" or is a delimiter. The P'.' Picture support needs a list to determine what characters are considered "invalid".

Both these functions use the same syntax to define the character string. The syntax is structured to allow this to be done in a simple manner.

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 "Only English A-Z and a-z are considered alphabetic" Select this if you do **NOT** 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

Setting the Invalid Character string for P'.'

To set the Invalid Character String:

- Enter OPTIONS as a Primary command. On the General Tab, you will see near the bottom a Text box labeled "Enter the Invalid characters for P'.' literals". Enter your desired invalid character list using the Syntax below.
- Click the Done button to save your changes.

Character String Syntax

The character 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 ÿ 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 string 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 **when specifying a WORD character string only**, not for an Invalid character string. 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.

If you wish to specifically exclude these international characters from these ranges, code the PROFILE WORD string like this:

A-Y Z a-y z

Note that the default WORD string used by SPFLite is: **A-Z a-z 0-9**,
the default for an Invalid Character string is: **00-1F 7F 81 8D 8F 9D**

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 is that all these strings must be specified using 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 ranges for EBCDIC files. Otherwise, choose these values carefully.

Word Processing Support

Contents of Article

[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 **MODE WORDS** or **MODE CHARS** to temporarily change this assumption. See the [MODE](#) 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 [MODE](#) 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 **aaabbb** 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 **aaabbb** 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.

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 a 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 CFG configuration 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 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 knows 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.

Created with the Personal Edition of HelpNDoc: [Keep Your PDFs Safe from Unauthorized Access with These Security Measures](#)

Up Down Left Right Top Bottom

The standard keyboard functions UP, DOWN, LEFT, RIGHT, TOP and BOTTOM are described fully in [Scrolling The Text](#).

Created with the Personal Edition of HelpNDoc: [Full-featured multi-format Help generator](#)

XFORM File transform macros

Contents of Article

Working with File Transforms and XFORM Macros

- [Basic concepts of Files Transforms](#)
- [Enabling and disabling the use of XFORM macros](#)
- [Maintaining the integrity of the XFORM session](#)
- [The XFORM READ action](#)
- [The XFORM WRITE action](#)
- [Impact of XFORM on STATE information](#)
- [Using XFORM for validation of file changes](#)
- [Importing and exporting external data](#)
- [Using XFORM with View, Browse or Read-Only files](#)
- [Using external programs to process XFORM data](#)
- [A cautionary note on using XFORM](#)
- [Sample XFORM macros](#)

Introduction

SPFLite has an advanced editing concept: **File Transforms and XFORM Macros**. This section will explain the concepts of File Transforms, and introduces XFORM macros, which implement the needed support.

A full explanation of XFORM macros and how to write and use them can be found in [A Tutorial on writing XFORM Macros](#).

This is a new concept, and while it is believed to satisfy the needed requirements, not all possible use cases have been explored or tested. If you feel changes or improvements would enhance it, be sure to report them on the SPFLite user forum at <https://spflite.freeforums.net/> so we can try to resolve the issue.

As this is new technology, it must be stressed that, until you are comfortable with this technique, and you have fully tested your XFORM macro(s), you should ensure you have adequate backups, or operate in "test mode" with copies of your original data. That sounds a lot like being in "beta test mode", and at first, since the XFORM macros are your creation, that's just what it will be.

A *file transform* is the process, by which the format of a file is copied and *temporarily changed* during the loading of a file, to making viewing and editing of the file easier, or to make it possible to view and edit a file that otherwise could not be edited at all. If a SAVE is issued, the *file transform* process is reversed, and the data is written back to the original file.

The primary reasons to use a file transform are:

- to access files in ways you ordinarily wouldn't or couldn't do

- to access files that cannot be directly edited by SPFLite

There are many reasons why editing some files with SPFLite may be inconvenient:

- Some files may be highly structured and technical.
- The file may not be 'readable' by normal means.
- The data may be a complex format, or need conversion to be editable.
- The file may include check sums or has encrypted or password-protected data.
- You might also need to look at, and possibly modify, files that are not "text" at all, such as EXE and DLL files.

SPFLite alone cannot handle such editing requirements. To address these complex types of edits, something more is needed. That is what a File Transform is for.

Basic concepts of File Transforms

To understand where File Transform and XFORM 'fit in' to augment SPFLite normal processing, it is helpful to consider what happens during an ordinary edit session.

During File OPEN

- User selects a file for EDIT, using an EDIT primary command or from a File Manager screen
- SPFLite starts up an edit session
- SPFLite reads the file
- SPFLite stores the file into memory
- The lines of data are made available for editing

During File SAVE

- SPFLite writes the lines in the edit session to the external file
- If STATE ON is in effect, STATE data, such as labels, tags, excluded-line markers, color highlighting, and other "attributes" of the file are written to the STATE data for this file.

A File Transform alters the Standard READ and Standard WRITE actions of an edit session, as discussed above. The items highlighted above will be performed by your XFORM macro.

An XFORM macro, like all macros in SPFLite, is written in thinBasic, and is stored in the SPFLite MACROS directory. It may have any name you wish.

During File Transforms, your XFORM macro will be replacing those standard READ and WRITE actions, *SPFLite itself will perform NO file I/O operations*. As such, your XFORM macro must utilize several of the file-based functions provided by thinBasic. To find the documentation you need for this, do the following:

- On an SPFLite command line, type **HELP BASIC**. This will bring up the Help file for thinBasic
- Scroll down the index on the left side of the document, until you see a major heading of **ThinBasic Modules**. (Don't go to ThinBasic Functions / File functions. That's the wrong part of the document.)
- Expand the ThinBasic Modules heading, and scroll down to the topic **File**. On the File topic, click on [+] if present to expand it.
- ThinBasic is very modular, before using the functions in any of the optional modules, you must tell thinBasic which you will be using. This is done with a **USES** command. So you need to insert a **USES "File"** statement in your XFORM macro. Then, you can

use whatever functions you require that are provided in the FILE module, such as OPEN, CLOSE, READ, WRITE, GET ATTRIBUTES, and so on.

A good naming convention for XFORM macros is to prefix the name with something unique, such as **X** or **X_** so they are easy to find in the MACROS directory when you create and edit them. The sample XFORM macros supplied with SPFLite have a prefix of **X**, such as XHEX.MACRO.

Enabling and disabling the use of XFORM macros

XFORM processing is optional, and is quite flexible in how you may or may not want it invoked. You can, for a particular file-type:

- Specify all XFORM activity as disabled. This is the default.
- Have a particular XFORM macro used for all I/O activity (**ON** mode).
- Specify an XFORM macro name, but leave it in dormant (or **OFF** mode).
- Override the normal XFORM setting for a single Edit. The override can:
 - Override the specified XFORM name with an alternate.
 - De-activate an XFORM which would normally be invoked. (Force **OFF**)
 - Activate an XFORM which was established in 'dormant (**OFF**) mode. (Force **ON**)
 - Activate a specific XFORM for a file which normally doesn't use XFORM.

To specify the use of an XFORM macro by changing a file's PROFILE setting, you can do one of the following:

- Issue a **PROFILE EDIT** while editing the type of file to which you wish to add XFORM support.
- Issue **PROFILE EDIT extension-name** from a command line to add XFORM support to a specific file type.
- In File Manager, click on **Configs**, which brings up a list of all profiles. Click on the profile name of interest, which will bring up the Profile Editor dialog.
- In an edit session, use the **XFORM** primary command to set, modify or display the XFORM settings for the file type of the file you are editing.

The **XFORM** command has the following syntax:

XFORM [*xname* | NONE] [**ON** | **OFF**]

If XFORM is entered with no parameters, it will simply display the current XFORM settings. The *xname* is the name of the macro to be used, where the macro is in the standard MACROS directory, with a file name of *xname*.MACRO.

The default for all file types is initially XFORM NONE. When you specify an *xname*, SPFLite will check to ensure the *xname*.MACRO exists, and if it doesn't, it will issue a warning. You can still make the setting; this is just a reminder for you to eventually define the macro when you have a chance.

The ON and OFF settings allow you to specify whether XFORM processing is to be used **all or most** of the time (when you'd use **ON**) or when XFORM will be used **seldom if ever** (when you'd use **OFF**). The ON and OFF settings can always be overridden for specific Edits (see below).

To override the normal XFORM settings on an EDIT (or similar) primary command, use an **overriding XFORM parameter**. This parameter may be one of the following, and must begin with a / slash:

- **/xname** - used to enable an XFORM macro named *xname*. This will override any

xname that might be defined in the profile. Use the `/xname` option for a file that rarely uses XFORM processing, or when you want to override an existing *xname* that might be in effect.

-
- **/NONE or /OFF** - used to disable XFORM processing, regardless of how the XFORM Profile option is set.
-
- **/ON** - used to enable XFORM processing. For this to work, the XFORM Profile option must already have an *xname* defined. It is an error to use **/ON** if the XFORM profile has an *xname* of NONE, since SPFLite will not know what macro to use.

Maintaining the integrity of the XFORM session

An XFORM edit is an unusual process, SPFLite is yielding control of the READ and WRITE processes to you, something that has not been supported before now. This requires you to cooperate by using the support appropriately. i.e. Follow the recommended procedures:

- If you are in the middle of editing a file using XFORM, don't change the XFORM settings prior to saving the file. Wait until you have saved it, before changing its XFORM settings.
- If you issue a SAVEAS, SPFLite changes the active filename of the current session. If this change also altered the file type, then the Profile for the new type will be loaded. If that Profile contains different XFORM settings. This opens the possibility for a variety of conflicts. In simple terms – **don't do this!**
- Likewise, if you issue a RENAME primary command, don't change the file extension on the file name, for the same reasons as for SAVEAS.
- Eventually, SPFLite *might* be able to carefully inspect the effects of a SAVEAS or RENAME and figure out the appropriate action to take. That is a long-term development effort, and still may not be able to resolve all issues. So for now, please understand that you have to "work within the system" for everything to function properly.

The XFORM READ action

The "XFORM READ" action operates as follows:

- User selects a file for EDIT, using an EDIT command or from a File Manager screen, ensuring that an XFORM *xname* is defined in the Profile, or supplied for the edit by an overriding XFORM option
- SPFLite starts up an edit session
- SPFLite launches the XFORM macro, providing it with the user-selected file name, and a parameter of "READ" to identify this as an "XFORM READ" action
 - The XFORM macro opens and reads the file, using thinBasic FILE operations
 - The XFORM macro re-formats the data however it wishes
 - The XFORM macro stores the reformatted data as text lines in the edit session
 - The XFORM macro closes the file, using thinBasic FILE operations
 - The XFORM macro passes back a return code indicating completion of the READ action
- The lines of data are made available for editing

Keep in mind that when the description above says things like, "the XFORM macro reads the file", it means that is what SPFLite *expects* it to do. What the macro *actually* does is entirely

up to the macro writer, so long as it load records into the Edit session.

A clever user *might* use the XFORM protocols to use XFORM macros for something *totally unrelated* to File Transforms. As long as you obey the protocols, what you actually *do* in the macro is up to you. It's hard to imagine what that might be, but SPFLite has some pretty smart users out there. One of these days, those users just might surprise us.

The XFORM WRITE action

The "XFORM WRITE" action operates as follows:

- The XFORM macro opens the external file for output, using thinBasic FILE operations
- The XFORM macro scans the data lines in the edit session, assembles and re-formats them as required, and the data is written to the external file on disk
- STATE data for the file is **NOT** written to the STATE data for this file, whether STATE ON is in effect or not. (See *Impact of XFORM on STATE information*, below.)
- The XFORM macro closes the file, using thinBasic FILE operations
- The XFORM macro passes back a return code indicating completion of the WRITE action
 - If the "XFORM WRITE action" was initiated by an END command, the edit session is closed and the tab it was located in is dismissed.

Impact of XFORM on STATE information

An XFORM macro does its own Read and Write operations, using functions built into the thinBasic script language, rather than having SPFLite do it. That is the whole point of a File Transform. So when an XFORM WRITE action occurs, SPFLite has no opportunity to "jump in" and save the file's STATE data. Thus any STATE information that might have existed beforehand is considered to be *invalidated*. There is no other way this can be handled. The old STATE data can no longer be relied upon.

If you are going to edit files using XFORM, and the macro saves the data, you are going to lose any STATE data that might have existed beforehand. Because of this limitation, most users are *probably* going to operate with STATE OFF for files processed via XFORM.

SPFLite decides how to handle this situation by examining the return code provided by the XFORM macro in it's HALT statement. The return code must be one of the following:

- **0**: XFORM WRITE action was successful, the file has been written to disk. SPFLite will delete any existing STATE information for the file. If this is a simple completion, you can set the message text of the HALT statement to a "" null string, or blanks. to prevent a message from getting displayed on the screen.
- **8**: XFORM WRITE action failed for some reason. The exact reason should be contained in the message text of the HALT statement. SPFLite will not delete any existing STATE information, and if the SAVE was triggered by END, it will not close the session.

Using XFORM for validation of file changes

Suppose you are using XFORM to edit some file that has a complex, strict data format which must be maintained, yet you want to be able to edit it. As the data in the Edit session has to be re-processed into the original data format, the XFORM WRITE function must obviously validate the data during the WRITE process. If an error is detected, the WRITE function

should format an appropriate message and issue it via a **HALT(8, error-message)**

This will signal SPFLite that the file has not been written, and it will return to normal Edit mode and display the error message. The user can correct the condition, and re-issue the END/SAVE command.

The extent of this validation is entirely up to the XFORM macro writer. It should be thorough to prevent saving bad data back to a file and possibly destroying it.

Importing and exporting external data

Various other SPFLite commands move data in and out of a session to external files. Commands such as COPY, CUT, PASTE, CREATE, REPLACE. For all such commands, an active XFORM macro is NOT INVOKED to perform the WRITE, all I/O is handled normally by SPFLite.

XFORM READ is only invoked for EDIT, BROWSE, VIEW and RELOAD.

XFORM WRITE is only invoked for SAVEAS and SAVE/END.

If you attempt to do one of these other WRITE actions, SPFLite will issue a message notifying you that XFORM is active, and the copy will be done without invoking the XFORM macro. It does that just as a reminder, and does not constitute an "error" condition. The message will give you an opportunity to cancel the command in case it was not what you intended.

Because COPY, PASTE, CREATE and REPLACE work this way, you can think of the text data you are working on as "the" file. The text becomes your "interface" to the real data. Being able to export and import data in this way allows you to conveniently save "pieces" of the original file in an easy and much more manageable, text-oriented way, rather than the possible-binary format of the original file. You may use this fact to devise clever and innovative ways to manage your data.

Using XFORM with View, Browse or Read-Only files

While an XFORM macro can be used to read a file into a VIEW or BROWSE session (or the file could simply be Read-Only), the XFORM WRITE function will never be invoked since SPFLite does not allow SAVE commands to be issued in that type of session.

Using external programs to process XFORM data

thinBasic, the script engine used by SPFLite, is remarkably fast and efficient. If your XFORM-processed files are not excessively large, and the transformation processing not overly complicated, the performance of XFORM macros should be more than sufficient for most users.

However, you may choose to, or need to process the data with an external utility application.

To do that, in your XFORM macro, you would use the SPFLite supplied functions SPF_Exec and/or SPF_Shell to launch the program(s) you needed. This would probably involve using temporary files to pass data between the external programs and your XFORM macro. But your XFORM macro will still ultimately end up passing text records into the Edit session.

A cautionary note on using XFORM

Advanced users might try to use the XFORM technology to access files they normally can't access, such as EXE and DLL files. If you do this merely to *look* at those files, that is fine.

However, if you create an XFORM procedure to *modify* such binary files, there is a saying for that: "Be careful what you wish for; you might get it."

Modifying such files requires detailed technical knowledge of their structure. The best advice we can give you is, *be sure you know what you're doing*. If you don't, you could destroy those binary files. Even if the change you intended is seemingly saved "successfully", you may still run into problems.

Sample XFORM macros

SPFLite supplies several sample XFORM macros, to help give you a guide as to writing your own, and you might wish to use some of them as-is if you find them useful. As samples, they illustrate techniques, useful functions you might need, and so on. There is no claim that these are the only way or most optimal way to accomplish the task at hand.

These macros represent the known "best practices" at the time they were written. Since XFORM is a new technology, these macros may be revised over time as XFORM support evolves. To ensure you are up to date with the latest techniques, come back frequently to the SPFLite user forum, ask questions and offer suggestions if you have them, and keep current with new SPFLite releases.

Here are the currently defined sample XFORM macros:

XCFG.MACRO	Shows access to a non-standard file type (SQL). Intended to read the Profile Names from the SPFLite CFG file
Xhex.MACRO	Provides a basic Hex file editor
Xlines.MACRO	Produces a dump format display of text lines. Demonstrates how to deblock RECFM U F and V format files
XRW.MACRO	Provides Read / Write access to a normal ANSI Text file
Xzip.MACRO	Shows access to a non-standard file type (ZIP). Produces a list of ZIP file contents

Created with the Personal Edition of HelpNDoc: [Produce Kindle eBooks easily](#)

Line Commands

Descriptions of the following Line Commands are contained in this section:

A	After destination
AA	After block
B	Before destination
BB	Before block
BNDS	Display BOUNDS line
C	Copy line(s)
CC	Copy a block
COLS	Display Columns line
D	Delete line(s)
DD	Delete a block
F	Display first lines in excluded region
G	Glue lines together

GG	Glue lines together in block
H	Here-destination lines
HH	Here-destination block
I	Insert temporary new line(s)
J	Join lines together
JJ	Join lines together in block
L	Display last lines in excluded region
LC	Lower-case lines
LCC	Lower-case lines in block
M	Move lines
MM	Move lines in block
MARK	Set column markers
MASK	Set the Insert line model
MD	Make data lines
MN	Make NOTE lines
N	Insert permanent new line(s)
NOTE / xNOTE	Insert NOTE lines
O	Overlay lines
OO	Overlay lines in block
OR	Overlay-Replace lines
ORR	Overlay-Replace lines in block
PL	Pad lines
PLL	Pad lines in block
R	Repeat lines
RR	Repeat lines in block
S	Show lines
SC	Sentence-case lines
SCC	Sentence-case lines in block
T	Select text lines
TT	Select text lines in block
TABS	Display Tabs line
TB	Text Break a line
TBB	Text Break lines in block
TC	Title-case lines
TCC	Title-case lines in block
TF	Text-flow a paragraph
TFF	Text-flow a block of paragraphs
TG	Text glue lines together
TGG	Text glue a block together
TJ	Text join lines together
TJJ	Text join a block together
TL	Trim lines
TLL	Trim lines in block
TM	Set Text Margin in a paragraph
TMM	Set Text Margin in a block of paragraphs
TR	Truncate lines
TRR	Truncate lines in block
TS	Text split a line

TU	Toggle User Line state of lines
TUU	Toggle User Line state of block
TX	Toggle excluded state of lines
TXX	Toggle excluded state of block
UC	Upper-case lines
UCC	Upper-case lines in block
U	Mark User lines
UU	Mark User lines in block
V	Revoke User line status
VV	Revoke User line status in block
W	Swap lines
WW	Swap lines in block
WORD	Display valid WORD characters
X	Exclude lines
XX	Exclude lines in block
(Column shift left
((Column shift left in block
)	Column shift right
))	Column shift right in block
≤	Data shift left
≤≤	Data shift left in block
≥	Data shift right
≥≥	Data shift right in block
[Indent shift left
[[Indent shift in block
]	Indent shift right
]]	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. 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 will result in a warning popup message advising you 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

[*Standard command forms with modifiers*](#)

[*Retain line-command modifier*](#)

[*Forward and backward modifiers*](#)

[*Post-exclude and post-unexclude 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](#), [O](#), [OO](#), [OR](#), and [ORR](#)

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 set an appropriate UNDO levels in [OPTIONS -> General](#) to enable the **UNDO** command to work as you need it to.

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 **/**
- a comma (,) in place of a ****
- 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](#), [\(](#), [\(\(](#), [\)](#), [\)\)](#)

Examples

Copy 3 lines after another and exclude the copied lines following the operation.

```
C3      AAA.....
000002 BBB.....
A-      CCC.....
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       CCC.....
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, the default is 1.

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](#)).
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 [<i>n</i>] / 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, the default is 1.

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.

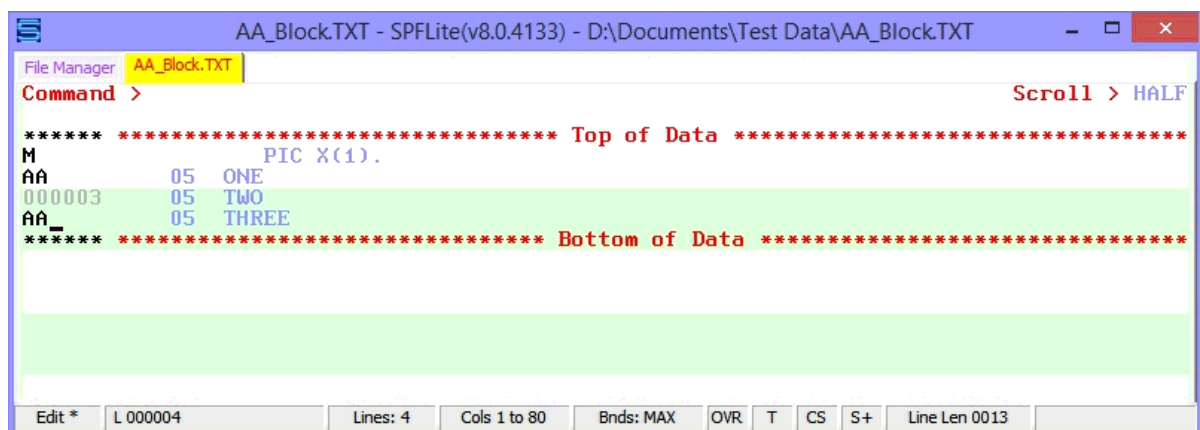
If a number **n** follows the **AA**, it means to copy after every *nth* line. If **n** is omitted it defaults to a value of 1, meaning 'after every line'.

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:



After:

AA_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\AA_Block.TXT

File Manager: AA_Block.TXT

Command: > Scroll > HALF

```

***** ***** 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 2**n**”, “relative line 3**n**”, 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: > Scroll > HALF

```

***** ***** Top of Data *****
M
AA2 *-----
000003 05 ONE
000004      PIC X(1).
000005 05 TWO
000006      PIC X(1).
AA_ *-----
000006 05 THREE
000007      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: > Scroll > HALF

```

***** ***** 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, the default is 1.

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 [<i>n</i>] / 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, the default is 1.

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.

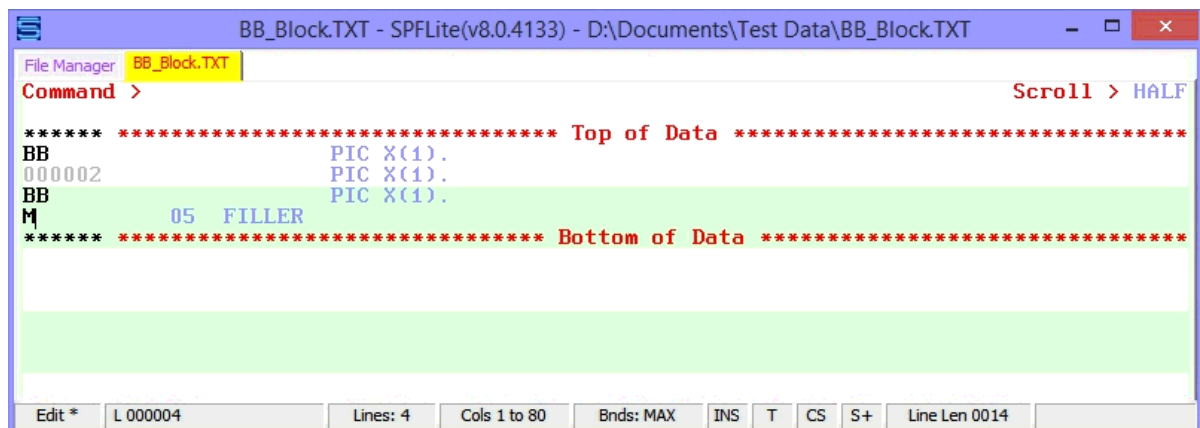
If a number **n** follows the **BB**, it means to copy before every *nth* line. If **n** is omitted it defaults to a value of 1, meaning before every line'.

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:



The screenshot shows the SPFLite editor window titled "BB_Block.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\BB_Block.TXT". The editor displays a command line with "BB" and a block of data lines. The data lines are highlighted in green. The command line is "BB" followed by "000002" and "M 05 FILLER". The data lines are "PIC X(1).", "PIC X(1).", and "PIC X(1).". The editor also shows a status bar at the bottom with fields for "Edit *", "L 000004", "Lines: 4", "Cols 1 to 80", "Bnds: MAX", "INS", "T", "CS", "S+", and "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 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 2**n**”, “relative line 3**n**”, 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
File Manager BB_Block.TXT
Command > Scroll > HALF
***** Top of Data *****
BB2| 000001 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 EFT specification).

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. If no > marker is specified, it is taken to mean MAX for the right boundary.
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 > 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

<code>C[n]</code> <code>CC / CC[n]</code>
--

Operands

n The number of lines to be copied. If you do not type a number, the default is 1.

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

<code>D [n]</code> <code>DD / DD</code>
--

Operands

n The number of lines to be deleted. If you do not type a number, the default is 1.

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, the default is 1.

Note that the / forward 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](#), and [S](#) line commands for alternative re-display commands.

G / GG - Glue Lines Together

Syntax

<code>G[n]</code> <code>GG / GG</code>

Operands

n The number of lines to be glued. If you do not type a number, the default is 1.

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 >
Scroll > HALF
Glued with "----"

***** ***** 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 Specification

Syntax

<code>H[n]</code> <code>HH[n] / HH</code>
--

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, the default is 1.

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.

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 NewTemporary Blank Lines

Syntax

<code>I [n]</code>

Operands

n The number of blank lines to insert. If you do not type a number, the default is 1.

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 model](#) for more information about defining a mask.

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 new information in the line, the space for the new line is automatically deleted the next time you press Enter.

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.

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, the default is 1.

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.

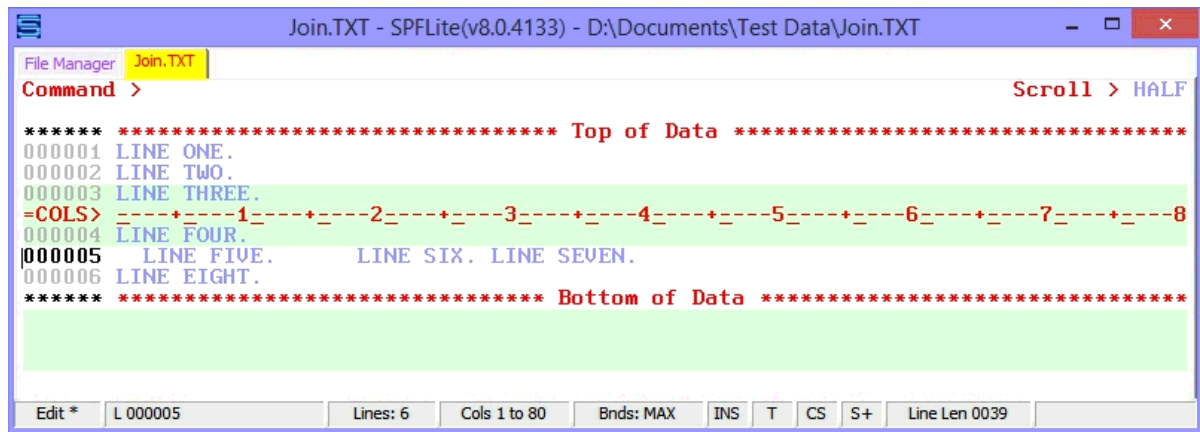
Example 1: Before Join,

```

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.
JJ LINE FIVE.
000006 LINE SIX.
JJ 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 Join:



The screenshot shows the SPFLite3 application window titled "Join.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Join.TXT". The window has a menu bar with "File Manager" and "Join.TXT". Below the menu bar is a "Command" prompt with a red ">" symbol. The main text area displays the following content:

```
***** ***** 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 *****
```

The text is displayed in a monospaced font. The line numbers are in blue, and the line text is in black. The "Top of Data" and "Bottom of Data" markers are in red. The "COLS" command is in red, and the column markers are in black. The text area has a light green background. At the bottom of the window is a status bar with the following fields:

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, the default is 1. |
|----------|---|
- 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](#), and [S](#) line commands for alternative re-display commands.

LC / LCC - LoweCase Lines

Syntax

LC [n] LCC / LCC	LCC is the block form of this command.
-----------------------------------	--

Operands

n The number of lines to be lower-cased. If you do not type a number, the default is 1.

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 [<i>n</i>] MM / MM [<i>n</i>]

Operands

n The number of lines to be moved. If you do not type a number, the default is 1. For the **MM** command, *n* is the number of copies of the **MM** block to be moved.

The *n* operand cannot be used on the **MM** command if it is participating in a line swap with a **W/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 **W/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:

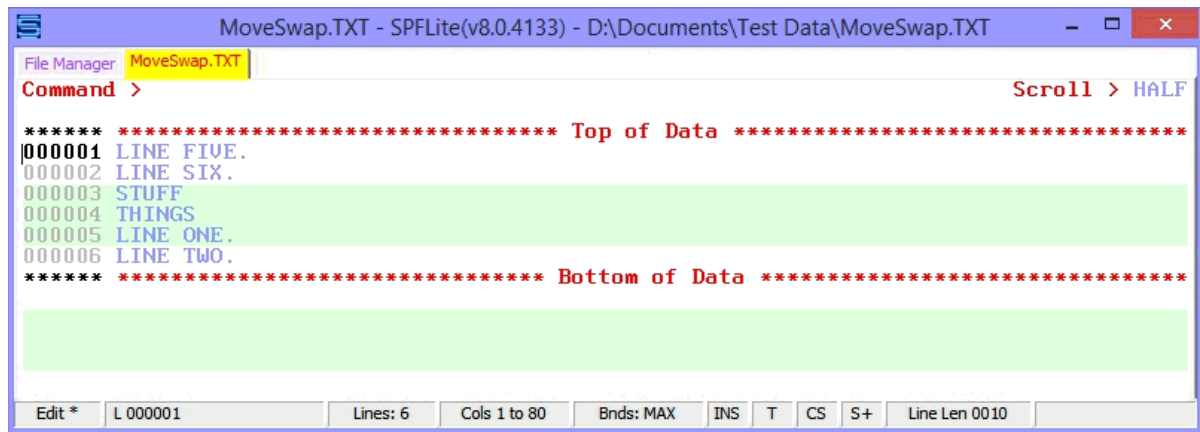
```

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 **>|<**.

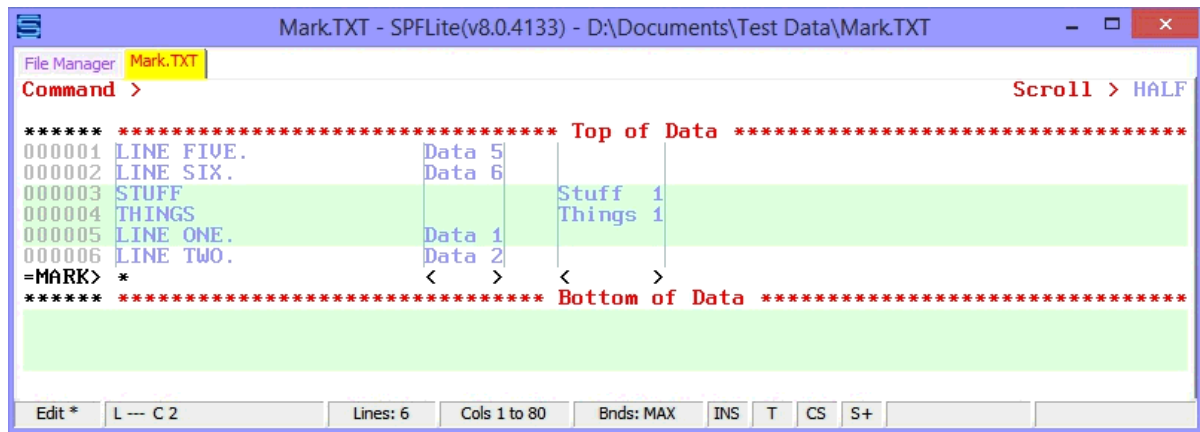
The ***** code from prior versions of SPFLite remains available, and works the same as the **<** code.

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:



The screenshot shows the SPFLite3 application window titled "Mark.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Mark.TXT". The window has a menu bar with "File Manager" and "Mark.TXT". Below the menu bar is a "Command" field with a right arrow. The main text area displays a table of data with a light green background. The table has four columns: a line number column, a text column, a "Data" column, and a "Stuff" column. The data is as follows:

Line	Text	Data	Stuff
000001	LINE FIVE.	Data 5	
000002	LINE SIX.	Data 6	
000003	STUFF		Stuff 1
000004	THINGS		Things 1
000005	LINE ONE.	Data 1	
000006	LINE TWO.	Data 2	

Below the table, there is a prompt "=MARK> *" and a "Bottom of Data" indicator. The status bar at the bottom shows "Edit *", "L --- C 2", "Lines: 6", "Cols 1 to 80", "Bnds: MAX", "INS", "T", "CS", "S+", and a scroll bar.

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 *a/ways* 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.

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.

Note: You may include symbolic variables on the line, both SPFLite **=xxx** style variables, and normal Windows style variables **%yyy%**. The substitution process will attempt to keep the location of the substituted variable aligned with the left-hand position of the symbolic variable. i.e. it will try to maintain column positioning whenever possible.

Example:

Using SPFLite variables AAA = Hello and BBBB = Yes

Model =BBBB TEXTA =AAA TEXTB

Would be substituted as follows, note the maintained column positions.

Model Yes TEXTA Hello TEXTB

- 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:

The screenshot shows the SPFLite3 editor window titled "MaskSample.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MaskSample.TXT". The editor displays a command line with "Command >" and a scroll bar on the right. The main text area shows the following content:

```
***** ***** Top of Data *****
000001 Data Line 1
000002 Data Line 2
000003 Data Line 3
000004 /*                                     */
000005 /*                                     */
000006 /*                                     */
000007 Data Line 4
000008 Data Line 5
000009 Data Line 6
***** ***** Bottom of Data *****
```

The status bar at the bottom shows "Edit *", "L 000004 C 1", "Lines: 9", "Cols 1 to 80", "Bnds: MAX", "INS", "T", "CS", "S+", and "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, the default is 1. 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:

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 *****
MD  PROFILE TXT UNLOCKED, AUTOBKUP OFF, AUTOCAPS ON, AUTOSAVE OFF PROMPT
MD  CAPS OFF, CASE T, CHANGE CS, COLLATE ANSI, COLS OFF, EOL CRLF, FOLD OFF
MD  HEX OFF, HILITE FIND AUTO, LRECL 0, MARK ON, MINLEN 0, PAGE OFF
MD  PRESERVE ON, RECFM U, SCROLL HALF, SETUNDO 5, SOURCE ANSI, START FIRST
MD  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> < +
000001 Data Line 1
=NOTE> This is a Note line to be kept
MD_ This is a Note line to be changed to a Data line
000002 Data Line 2
***** ***** Bottom of Data *****

```

Edit * Lines: 2 Cols 1 to 80 Bnds: MAX OVR T CS S+

Result: Notice that the lines that had **MD** line commands placed on them now have line numbers, because they are now regular data lines.

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 *****
000001 PROFILE TXT UNLOCKED, AUTOBKUP OFF, AUTOCAPS ON, AUTOSAVE OFF PROMPT
000002 CAPS OFF, CASE T, CHANGE CS, COLLATE ANSI, COLS OFF, EOL CRLF, FOLD OFF
000003 HEX OFF, HILITE FIND AUTO, LRECL 0, MARK ON, MINLEN 0, PAGE OFF
000004 PRESERVE ON, RECFM U, SCROLL HALF, SETUNDO 5, SOURCE ANSI, START FIRST
000005 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
000008 Data Line 2
***** ***** Bottom of Data *****

```

Edit * L 000001 C 1 Lines: 8 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0071

MN / MNN - Make Note Line

Syntax

MN [<i>n</i>] MNN / MNN

Operands

n	The number of lines to be converted to note lines. If n is omitted, the default is 1. 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.

- 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:

The screenshot shows the SPFLite3 editor window titled "MD_Test.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MD_Test.TXT". The editor displays the following content:

```

***** ***** 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>
=MARK>
=MARK>
=MARK>
=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 *****

```

The status bar at the bottom shows: Edit *, L 000008, Lines: 8, Cols 1 to 80, Bnds: MAX, OVR, T, CS, S+, Line Len 0037.

Result: Notice that the lines that had MN line commands placed on them now have the **=NOTE>** marker, because they are now **NOTE** lines.

The screenshot shows the SPFLite3 editor window titled "MD_Test.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\MD_Test.TXT". The editor displays the following content:

```

***** ***** Top of Data *****
=NOTE> PROFILE TXT UNLOCKED, AUTOBKUP OFF, AUTOCAPS ON, AUTOSAVE OFF PROMPT
=NOTE> CAPS OFF, CASE T, CHANGE CS, COLLATE ANSI, COLS OFF, EOL CRLF, FOLD OFF
=NOTE> HEX OFF, HILITE FIND AUTO, LRECL 0, MARK ON, MINLEN 0, PAGE OFF
=NOTE> PRESERVE ON, RECFM U, SCROLL HALF, SETUNDO 5, SOURCE ANSI, START FIRST
=NOTE> STATE ON, TABS ON, XTABS 4
=WORD> A-Z a-z 0-9
=MARK>
=MARK>
=MARK>
=MARK>
=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
=NOTE> Data Line 2 - TO BE CHANGED TO A NOTE
***** ***** Bottom of Data *****

```

The status bar at the bottom shows: Edit *, L --- C 1, Lines: 2, Cols 1 to 80, Bnds: MAX, OVR, T, CS, S+.

N - Insert New Permanent Blank Lines

Syntax

N [<i>n</i>]

Operands

n	The number of permanent blank lines to insert. If you do not type a number, the default is 1.
----------	---

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 **I** 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

<code>NOTE [n] or xNOTE [n]</code>

Operands

n	The number of new, blank =NOTE> or xNOTE> lines to be created. If n is omitted, the default is 1.
----------	--

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. The last three note types (**WNOTE**, **XNOTE** and **YNOTE**) are treated as temporary Notes, that is, they will not be saved and restored by [STATE](#) saving.

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.

See [Working with NOTE and xNOTE Lines](#) for more information.

O / OO - Overlay Lines

Syntax

<code>O[n]</code> <code>OO / OO</code>

Operands

n	The number of lines to be overlaid. If you do not type a number, the default is 1.
----------	--

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 **M/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.

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:

The screenshot shows the SPFLite3 interface with the file 'Overlay.TXT' open. The command line shows 'Command >'. The text area displays the following content:

```

***** 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 other data lines
000014 other data lines
***** Bottom of Data *****

```

Lines 4-6 and 10-12 are highlighted in green. The status bar at the bottom shows 'Edit', 'L 000013', 'Lines: 14', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', and '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.

The screenshot shows the SPFLite3 interface with the file 'Overlay.TXT' open. The command line shows 'Command >'. The text area displays the following content:

```

***** Top of Data *****
000001 other data lines
000002 other data lines
----- < 000001 > -----
000004 A11
000005 A22
000006 A33
----- < 000001 > -----
000008 other data lines
----- < 000001 > -----
000010 B11
000011 B22
000012 B33
----- < 000001 > -----
000014 other data lines
***** Bottom of Data *****

```

Lines 4-6 and 10-12 are highlighted in green. Dashed lines with '< 000001 >' are placed next to lines 4, 6, 10, and 12. The status bar at the bottom shows 'Edit', 'L 000004 C 1', 'Lines: 14', 'Cols 1 to 80', 'Bnds: MAX', 'OVR', 'T', 'CS', 'S+', and 'Line Len 0003'.

Now, it is very obvious where to place the **OO** and **MM** line commands:

Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT

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:

Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT

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.

Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT

File Manager Overlay.TXT

Command > reset 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 * 2014-05-13 14:26:34 Lines: 11 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0007

Using Excluded Line as the Source and/or Target of an Overlay Operation

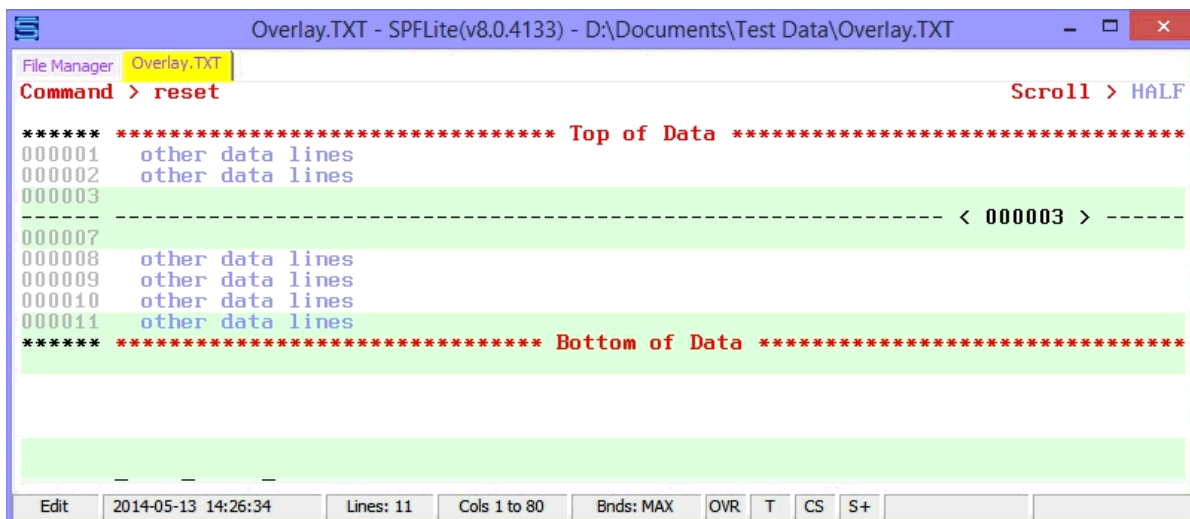
When you have a large number of lines involved in an Overlay operation, it may be convenient

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:

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):

Because both the source range and the target range were excluded, the target range remains excluded after the Overlay operation is completed:

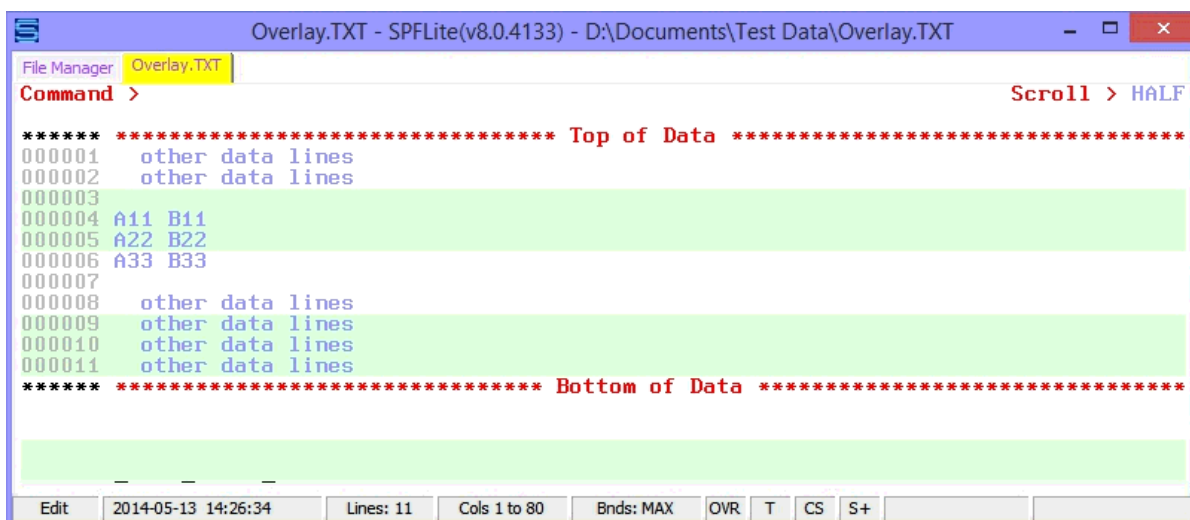


The screenshot shows the SPFLite3 application window titled "Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT". The "File Manager" tab is active, showing "Overlay.TXT". The "Command" field contains "> reset". The "Scroll" field shows "> HALF". The main display area shows a list of lines with addresses 000001 to 000011. Lines 000001 and 000002 are labeled "other data lines". Lines 000003 to 000007 are highlighted in green. Line 000003 contains "< 000003 >". Lines 000008 to 000011 are labeled "other data lines". The status bar at the bottom shows "Edit", "2014-05-13 14:26:34", "Lines: 11", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", and a scroll bar.

```
Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT
File Manager Overlay.TXT
Command > reset Scroll > HALF
***** 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 *****
Edit 2014-05-13 14:26:34 Lines: 11 Cols 1 to 80 Bnds: MAX OVR T CS S+
```

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.



The screenshot shows the SPFLite3 application window titled "Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT". The "File Manager" tab is active, showing "Overlay.TXT". The "Command" field contains ">". The "Scroll" field shows "> HALF". The main display area shows a list of lines with addresses 000001 to 000011. Lines 000001 and 000002 are labeled "other data lines". Lines 000003 to 000007 are highlighted in green. Line 000003 contains "< 000003 >". Lines 000008 to 000011 are labeled "other data lines". The status bar at the bottom shows "Edit", "2014-05-13 14:26:34", "Lines: 11", "Cols 1 to 80", "Bnds: MAX", "OVR", "T", "CS", "S+", and a scroll bar.

```
Overlay.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Overlay.TXT
File Manager Overlay.TXT
Command > Scroll > HALF
***** Top of Data *****
000001 other data lines
000002 other data lines
000003 < 000003 >
000004 A11 B11
000005 A22 B22
000006 A33 B33
000007
000008 other data lines
000009 other data lines
000010 other data lines
000011 other data lines
***** Bottom of Data *****
Edit 2014-05-13 14:26:34 Lines: 11 Cols 1 to 80 Bnds: MAX OVR T CS S+
```


OR / ORR - Overlay Replace Lines

Syntax

<code>OR [n]</code> <code>ORR / ORR</code>	ORR is the block form of this command.
---	--

Operands

n The number of lines to be overlay-replaced. If you do not type a number, the default is 1.

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 only blank characters in the receiving lines, on a character by character basis.

For Overlay-Replace command, the overlay is performed **regardless** of whether the position in the receiving line is blank or not.

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 [n] / PLL	PLL is the block form of this command.
--	--

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 [<i>n</i>] RR [<i>n</i>] / RR

Operands

n The number of lines to be repeated. If you do not type a number, the default is 1.

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 - Show lines

Syntax

S

Operands

None.

Description

To re-display an excluded line group simply type **S** in the line command area of the excluded line group and press Enter.

Notes:

1. If you are running in HIDE mode, and there **is** no line command area to enter the **S**, enter it on the normal line preceeding the hidden group. i.e. the line command area which is underlined.
2. Other than point 1. above, an **S** entered on a normal unexcluded line will have no effect.
3. If you are in Multi-Edit mode, and lines for a **FILE=>** are excluded, an **S** on the **FILE=>** line will also un-exclude the lines.
4. Also see the F and L line commands for alternative re-display commands.

SC / SCC - Sentence Case Lines

Syntax

SC [n] SCC / SCC	SCC is the block form of this command.
-----------------------------------	--

Operands

n The number of lines to be converted to Sentence Case. If you do not type a number, the default is 1.

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 (?).

T / TT - Select Text Lines

Syntax

<code>T[n]</code> <code>TT / TT</code>

Operands

n The number of lines to be selected. If you do not type a number, the default is 1. 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:

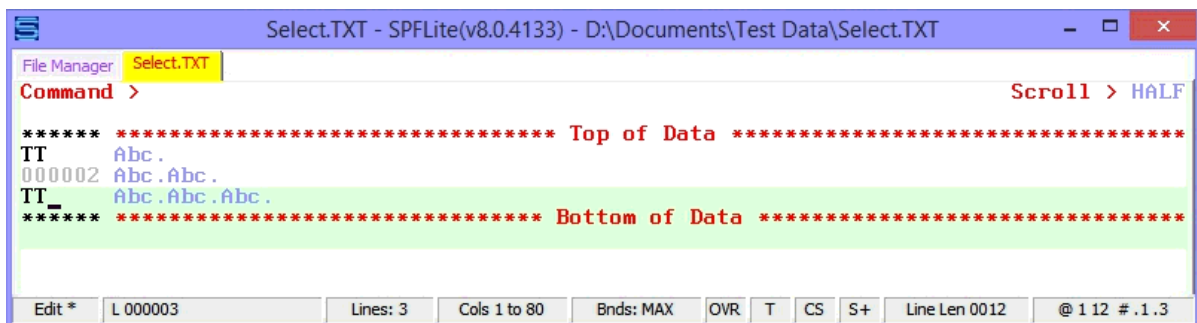
1. Type **T** in the line command area of the line that you want to select. If you want to select one or more lines that immediately follow this line, type a number greater than 1 immediately after the **T** command.
2. Press Enter.
3. The lines are selected.

To select a block of lines:

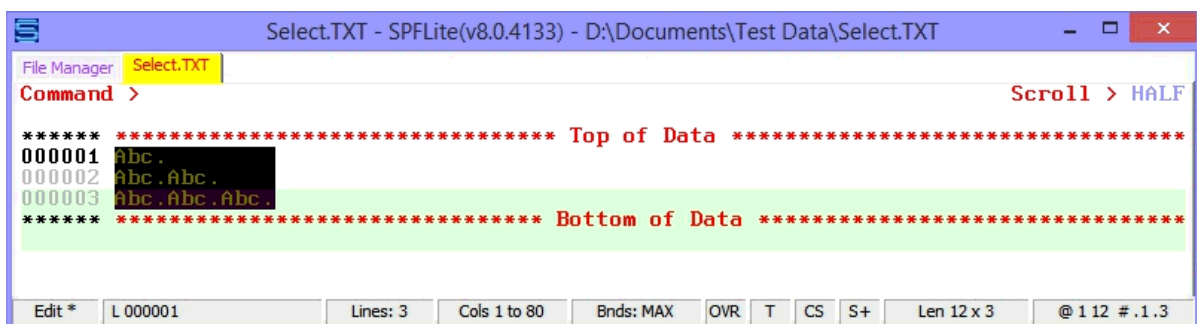
1. Type **TT** in the line command area of both the first and last lines that you want to select. You can scroll (or use [FIND](#) or [LOCATE](#)) between typing the first **TT** and the second **TT**, if necessary.
2. Press Enter
3. The lines that contain the two **TT** commands and all of the lines 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.

Select.TXT - SPFLite(v8.0.4133) - D:\Documents\Test Data\Select.TXT

File Manager Select.TXT

Command > Scroll > HALF

```
***** ***** Top of Data *****
000001      Abc.
000002      Abc.Abc.
000003  Abc.Abc.Abc.
***** ***** Bottom of Data *****
```

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).

See "[Defining and Using TABS](#)" for more information on using Tabs.

TB / TBB - Text Break Lines

Syntax

TB [n] TBB / TBB	TBB is the block form of this command.
---	--

n The number of permanent blank lines to be inserted between the broken lines. If you do not type a number, the default is 1.

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 / TCC	TCC is the block form of this command.
-----------------------------------	--

Operands

n The number of lines to be converted to Title Case. If you do not type a number, the default is 1.

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 / TFF	TFF is the block form of this command.
----------------------------------	--

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 in indent of 4 columns on the remaining lines.

The formatting is done this way 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 the 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 / TGG	TGG is the block form of this command.
----------------------------------	--

Operands

n The number of lines to be glued. If you do not type a number, the default is 1.

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:

```

***** 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.

```


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.
000005 LINE FIVE.
000006 LINE SIX.
000007 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 / TJJ	TJJ is the block form of this command.
----------------------------------	--

Operands

n The number of lines to be joined. If you do not type a number, the default is 1.

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.
Edit * L 000007 Lines: 8 Cols 1 to 80 Bnds: MAX OVR T CS S+ Line Len 0011
```

After 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.
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 0034
```

TL / TLL - Truncate Lines to a Length

Syntax

TL [n] TLL [n]	TLL is the block form of this command.
---	--

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

[Text Margin and BOUNDS](#)

[A note about using TF/TFF and TM/TMM to format text](#)

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 in 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 the 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.

TR / TRR - Trim Trailing Blanks

Syntax

TR[n] TRR / TRR	TRR is the block form of this command.
----------------------------------	--

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, the default is 1.
	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, the default is 1.

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 (NU lines), and lines that formerly were ordinary lines (NU 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 (NU 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 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

<code>U[n]</code> <code>UU / UU</code>

Operands

n The number of lines to be marked as User lines. If 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 "NU lines", where U lines ("User lines") are a set of one or more lines of special interest at some particular time, and NU lines are everything else. User lines are marked with a | vertical bar in the "gap column", and NU lines are unmarked. The unmarked NU lines are the same, ordinary data lines you have always worked with.

A file will ordinarily consist entirely of NU lines, which is the default state of a file. Other than having one or the other of these U/NU states, there is no difference between U and NU 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, the default is 1.

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

<code>V[n]</code> <code>VV / VV</code>

Operands

n	The number of lines whose User Line status is to revert back to that of ordinary lines (NU lines). If 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 "NU lines", where U lines ("User lines") are a set of one or more lines of special interest at some particular time, and NU lines are everything else. User lines are marked with a | vertical bar in the "gap column", and NU lines are unmarked. The unmarked NU lines are the same, ordinary data lines you have always worked with.

A file will ordinarily consist entirely of NU lines, which is the default state of a file. Other than having one or the other of these U/NU states, there is no difference between U and NU 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, the default is 1,, only the line on which you type W is swapped. For the WW command, the *n* operand cannot be used.

Description

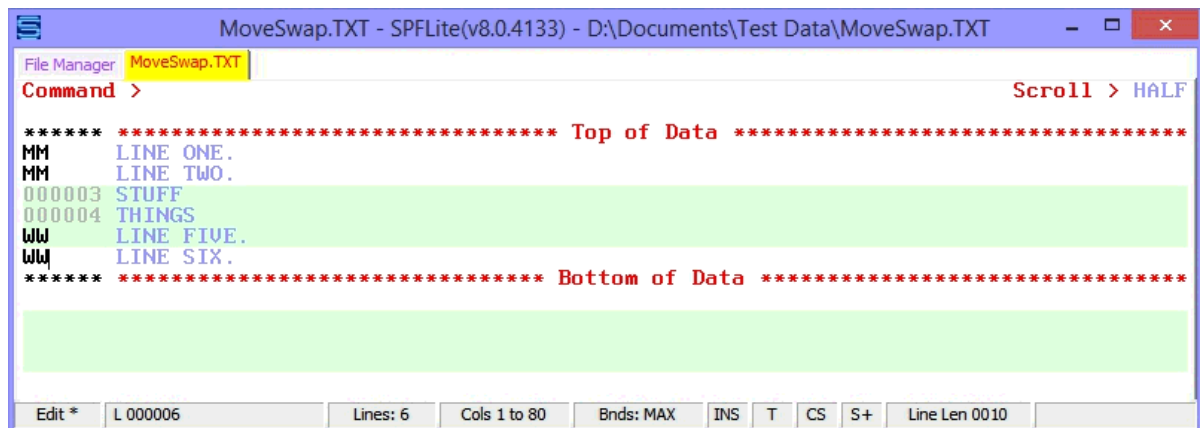
The W/WW line command is used to mark off one or more lines which are swapped with one or more lines marked with M or MM. W and WW are only valid for swapping lines between an M/MM block, and cannot be used for any other purpose.

See the description of M/MM for more information.

Note: See [Word Processing Support](#) for information about text swaps.

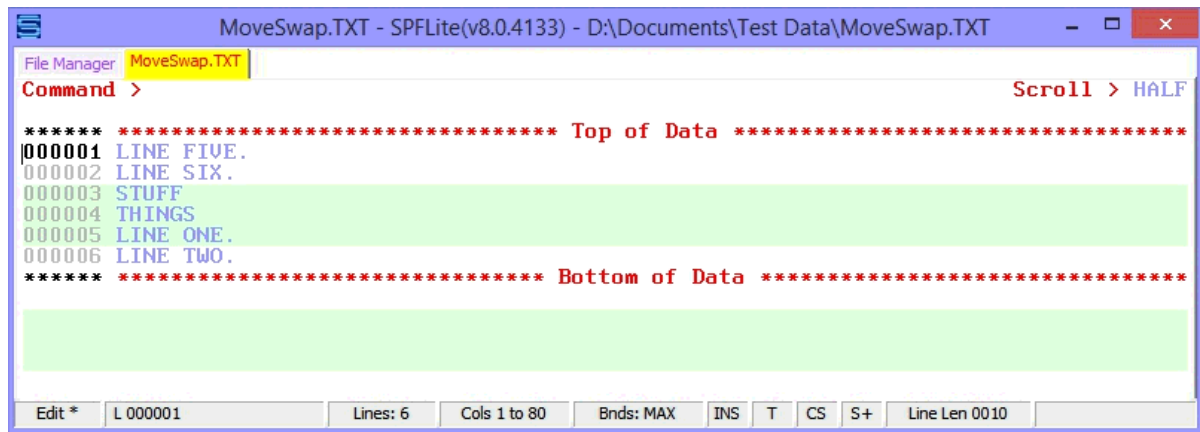
Example of using M/MM and W/WW to swap lines:

Before:



```
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

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.

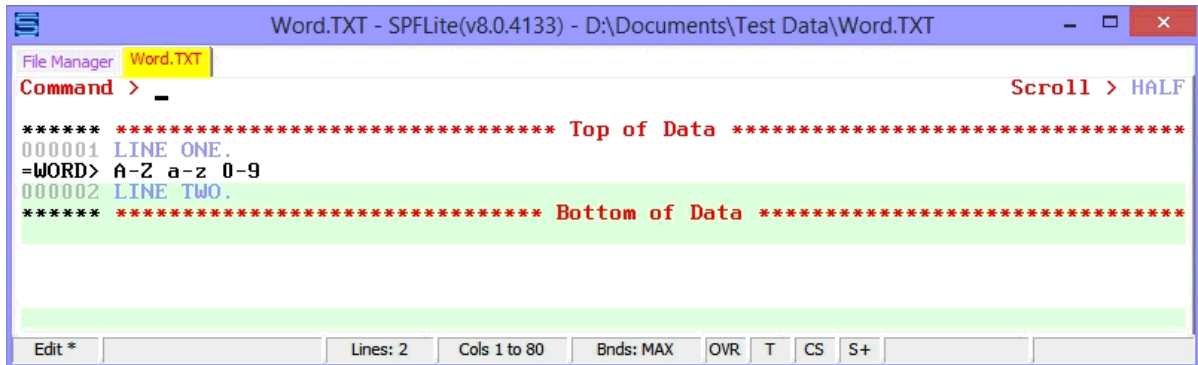
For example, if you are editing a "C" program file which allows underscores in names, the underscore is a letter-like character and not a delimiter. To successfully find "C" words in such files, you would **add** the underscore to the list of characters in the **WORD** line. Likewise, many IBM mainframe 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:



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](#).

"Note that the function automatically includes all international language characters which fall logically into the upper (A-Z) and lower (a-z) case character ranges. To exclude these characters from these ranges, code the PROFILE WORD string like this: A-Y Z a-y z

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, the default is 1.
----------	--

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 specified in [Options -> General](#).

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.

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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 specified in [Options -> General](#).

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

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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 specified in [Options -> General](#).

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.

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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 specified in [Options -> General](#).

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 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.

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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.

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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 *indent increments* 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.

It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

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.

Created with the Personal Edition of HelpNDoc: [Transform Your Help Documentation Process with a Help Authoring Tool](#)

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 be performed
ADD	Add a text line
ALIGN	Align string on a column Boundary
APPEND	Add text to end of lines
AUTOBKUP	Control backup creation
AUTOCAPS	Control auto-capitalization of language keywords
AUTONAME	Specify an alternate AUTO file name
AUTOSAVE	Control automatic file save defaults
BACKUP	Request a Backup of the current file
BOM	Set BOM writing ON / OFF
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
CLONE	Open an un-named edit using an existing file
CLS	Clear or Close the SPFLite Debug Window
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
DCB	Specify the RECFM, LRECL and EOL together.
DELETE	Delete selected lines
DIFF	Perform a DIFF file compare between two files
DIR	Display folder containing this file in File Manager
DO	Execute the contents of a DO file
DOWN	Scroll downward in the data
EDIT	Open a file for editing

EFT	Edit the Extended File Type Criteria
END	End the edit session
ENUMWITH	Change Increment for Enumerate Functions
EOL	Specify End-of-Line characters. Deprecated - Use DCB .
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
FLIP	Reverse exclusion status of lines
FORMAT	Specify file characteristics
GLUEWITH	Specify a join string for Glue operations
HELP	Display the Help file
HEX	Set HEX display mode ON or OFF
HIDE	Hide excluded lines
HILITE	Control text highlighting options
IMACRO	Specify a macro to be run immediately after file load
JOIN	Join lines using Find / Change strings
KEYMAP	Display keyboard settings dialog
LABEL	Search for a line and add a Line Label
LC	Lower-case a range of lines
LEFT	Scroll leftward in the data
LOCATE	Scroll the display to a specified line
LOOPCHECK	Turn loop detection ON or OFF
LRECL	Specify logical Record Length. Deprecated - Use DCB .
MACLIB	Specify a unique Macro library for a Profile
MAKELIST	Create a FILELIST from the current display in File Manager
MARK	Turn MARK lines ON or OFF
MEDIT	Add a file to a Multi-Edit session
MODE	Specify and set the session Edit mode
NDELETE	Delete lines where string is not found
NFIND	Find lines where string is not found
NFLIP	Reverse exclusion status of lines where string is not found
NEXCLUDE	Exclude lines where string is not found
NREVERT	Revert user line status when string not found
NSHOW	Show (unexclude) lines where string is not found
NULINE	Mark User line status when string not found
OPEN	Edit another file in a new session
OPTIONS	Set editor global options
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	Send selected lines to the printer
PROFILE	Display current file profile values
PTYPE	Enter PowerType mode
RCHANGE	Repeat change

RECALL	Recall (Open) a favorite FILELIST
RECFM	Specify Record Format. Deprecated - Use DCB .
REDO	Redo an UNDO action
RELOAD	Reload current edit file from disk
RENAME	Rename the current edit 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
SHOW	Show (unexclude) lines where a string is found
SORT	Sort the edit data
SOURCE	Specify character encoding
SPLIT	Split lines 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 lines as User lines
UNDO	Undo changes
UP	Scroll upward in the data
VIEW	View a file in read-only mode.
WDIR	Open Windows Explorer for this file folder.
XFORM	Specify a macro to handle external file reading / writing
XSUBMIT	Submit an external file
XTABS	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 they 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 select 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.

Optional Operands

Not all of the operands shown below are necessarily valid with all commands that specify line-range-operands. e.g. A SHOW command would not support X/NX operands because those attributes are an inherent part of the command's function.

Syntax

<pre>[{ .start-label .start-line-number } [{ .end-label .end-line-number }]] [:tag :\tag] (Where tag may be ZALL to indicate ANY tagname) [X NX] [U NU]</pre>	
<div>Alternative Line Command Specification - CC and MM Blocks</div> <p>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 M/MM line commands can be used to mark the lines to be used and then the primary command entered without a line range specification.</p>	
<div>Single character text selection operands</div> <p>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.</p>	

Operands

start-line-number	This should specify the starting line number of a range. This 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 .

end-line-number	This should specify the ending line number of a range. This 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 . NOTE: If no <i>end-line-number</i> or <i>end-label</i> are provided, the starting line number is treated as a one line range.
: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. You may specify :ZALL to indicate a match of ANY tagname/
:!tag	Requests processing of all lines which do not contain :tag . If you specify :ZALL it means any line which contains NO tag.
X	This requests the command only process lines which are Excluded (X)
NX	This requests the command only process lines which are NOT Excluded (NX)
U	This requests the command only process lines which are marked as User lines (U)
NU	This requests the command only process lines which are NOT marked as User lines (NU)

Description

The above operands may be combined to refine your selection.

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 labelled .AAA to the line labelled .BBB.

FIND "DEF" 3 30 .110 .120

Find string "DEF" in columns 3 to 30 on lines 110 thru 120.

FIND "GHI" .110 .120 :T

Find string "GHI" in lines tagged with :T, in lines 110 thru 120.

FIND "PQR" :QTAG

Find string "PQR" anywhere on any line currently tagged with the tag-name **:QTAG**.

Processing Order / Priority

When multiple selection operands on a command are chosen, it is important to keep the following in mind, especially so when using the Negative style search commands (NFIND, NEXCLUDE etc.) which can be confusing.

1. Line-range selection is always performed **first**. And a failure terminates the command.
2. Once a line passes line-range selection, and only then, will it be tested against any string comparison tests.

Normal positive tests

For normal tests, **ALL** entered selection operands are tested in an **AND** relationship, meaning the **First** failure terminates the command. To be successful, **ALL** tests are performed and must be successful. This is the normal expected handling.

Negative Tests

When performing Negative test commands, a failure in the *line-range* tests (1. above) will terminate the command, and any string comparison tests are not even performed. This can trigger unexpected results if you expected all operands to be evaluated.

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 -Highlights](#). (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 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 -Highlights](#). (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

Syntax

ACTION ['n' ?]

Operands

'n'	The desired interval between automatic SAVE. (a '0' zero value indicates no automatic saves are desired.
?	Display the current status of the setting.

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 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 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.

Note: The counting will only occur while the file is in *modified* status, as this is the only condition where you are exposed to loss of changes. Keep in mind that when an automatic SAVE does occur, the modified flag is reset, so further automatic saves will not occur until after the file is modified again.

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 Specifies 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
```

ALIGN - Align string on Column Boundary

Syntax

```
ALIGN      string-1
           [ PREFIX | SUFFIX | WORD | CHAR ]
           [ C ] [ Q ] [ T ]
           [StartScan-column] { align-column | MAX }
           [ line-control-range ]
           [ color-selection-criteria]
           [ DS | CS ]
           [ ALL ]
```

Operands

string-1 The string being searched for. The string should contain the single occurrence of the value you wish to shift right to an alignment-column. This string may be any normal search value, a normal literal, a Picture string, or a Regular Expression

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-Word characters.

CHAR Locates search-string regardless of what precedes or follows it.

C - Locate the search string **within** a defined Comment string.

Q - Locate the search string **within** a defined Quoted literal string.

T - Locate the search string **within** plain text (i.e. Not in a Comment or Quoted string).

You may enter more than 1 of **C** **Q** or **T** to customize the selection. They are tested in an **OR** relationship.

These three operands require a valid Profile with Colorization active.

StartScan-column If this optional column is provided, it identifies the starting column in each line at which the search for the string-1 is to begin.

align-column You may specify either:

- | MAX**
 - a specific desired column number to which you wish the string to be moved. If the found string is to the right of this column (and would thus be shifted left) no alignment will take place. A warning message will be produced advising you that this has occurred.
 - MAX** - which requests SPFLite to scan for all valid occurrences of the string-1 and use the right-most found column location as the desired alignment column.

line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
CS DS	Specifies the Column Shift / Data Shift mode for this particular ALIGN command. See Effect of CHANGE/ALIGN 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.
ALL	ALL requests that all lines in the line range will be aligned. If the ALL option is omitted, then by default only the first line in the lines in the line range will be aligned.

Description

ALIGN is used to simplify a common requirement. It provides a simple method to re-arrange or 'columnize' data. Your data may be in some kind of 'ragged' columnar format and manually shifting data line by line can be very tedious.

Alignment Samples

This example aligns the data based on the word 'AND'

```
Command > ALIGN 'AND' ALL 15
*****  ----+----1----+----2----+----3----+----4----+
000001  ABCDEF AND GHIJKL
000002  JKLMN  AND OPQRS
000003  Y  AND N
```

Result:

```
*****  ----+----1----+----2----+----3----+----4----+
000001  ABCDEF          AND GHIJKL
000002  JKLMN           AND OPQRS
000003  Y               AND N
```

This example is similar, but shows the use of a startscan-column operand

```
Command > ALIGN 'AND' ALL 5 15
*****  ----+----1----+----2----+----3----+----4----+
000001  ABC AND DEF
000002  JKLMN AND OPQRS
000003  Y AND N
```

Result:

```
*****  ----+----1----+----2----+----3----+----4----+
000001  ABC           AND DEF
000002  JKLMN         AND OPQRS
000003  Y AND N
```

Note: Line 000003 was skipped because the AND string was ignored since it was to the left of

the StartScan-column.

The next examples show the difference between using the CS and DS operands

Command > ALIGN 'AND' ALL 20 CS

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001  ABC      AND              DEF
000002  JKLMN AND                      OPQRS
000003  Y AND                                N
```

Result:

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001  ABC              AND          DEF
000002  JKLMN            AND          OPQRS
000003  Y                AND          N
```

Note the spacing between the AND and the following words is maintained since CS (Column Shift Mode) was in effect.

Command > ALIGN 'AND' ALL 20 DS

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001  ABC      AND              DEF
000002  JKLMN AND                      OPQRS
000003  Y AND                                N
```

Result:

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001  ABC              AND          DEF
000002  JKLMN            AND  OPQRS
000003  Y                AND  N
```

Note: Because DS (Data Shift Mode) was in effect, the results are dramatically different.

For line 000001 there are no 'spare' spaces between AND and the next word to allow shifting without also moving the DEF string, so the change is done in CS (Column shift mode)

For line 000002, there is plenty of unused space between AND and OPQRS, so the AND is shifted without moving the OPQRS string.

For line 000003, there is also plenty of spare room, so AND is moved, and N remains in place.

The next examples show how the MAX column request works.

Command > ALIGN 'AND' ALL MAX CS

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
000001  ABC      AND              DEF
000002  JKLMN AND                      OPQRS
000003  Y AND                                N
```

Result:

```
*****  ----+-----1-----+-----2-----+-----3-----+-----4-----+
```

```
000001 ABC      AND      DEF
000002 JKLMN    AND      OPQRS
000003 Y        AND      N
```

SPFLite scans the range and the rightmost **AND** is in line 00001, Column 8, so 8 is used as the alignment column.

Created with the Personal Edition of HelpNDoc: [Import and export Markdown documents](#)

ALL - Perform a Cmd. on all FM files

Syntax

ALL	[
(File	BROWSE/BRO/B
Manager)	CLONE/C
	CANCEL/CAN
	EDIT/E/SELECT/S
	END
	MEDIT/M
	PRINT/P
	TOUCH/T
	VIEW/V
]

Operands

command The normal File Manager Line Command which is to be performed against all files currently being displayed in File Manager.

Description

The **ALL** command provides a simple way to issue a File Manager line command against ALL the files currently displayed.

Example: **ALL MEDIT** would open all displayed files in a single [MEdit](#) session.

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

APPEND - Add text to end of line

Syntax

APPEND	string
	[line-control-range]
	[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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
ALL	All lines in the line range are 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 [ON OFF ?]

Operands

ON | **OFF** The desired AUTO-BACKUP status

? Display the current status of the setting.

Description

If no operand is entered, the command will 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

If **ON** is specified, SPFLite will create a backup copy of any Edited file by invoking [BACKUP](#) for the file under the following circumstances:

- A [BACKUP](#) will not be performed unless a [SAVE](#) (or a SAVE performed during [END](#) processing) is done.
- A [BACKUP](#) will be performed only **once** per edit session, regardless of how many [SAVE](#) commands are issued.

If **OFF** is specified, then no automatic Backups will be created.

The value is stored as part of the PROFILE options which are maintained individually by file type.

Further details of Backup can be found in ["Working with BACKUP & RESTORE"](#)

AUTOCAPS - Control AutoCaps Mode

Syntax

AUTOCAPS [ON OFF ?]

Operands

ON | **OFF** The desired AUTOCAPS status

? Display the current status of the setting.

Description

AUTOCAPS controls whether the capitalization done by [Auto Capitalization Support](#) is made permanent when the file is saved. The [Auto Capitalization Support](#) normally is, like colorization support, only temporary and only affects the display of the text on-screen.

But since it **is** possible to save capitalization to the written file, this option is made available to give you that choice. Note this is only possible if you have **HILITE AUTO ON** in the File Profile **and** the .AUTO file actually requests capitalization of defined keywords.

Operands

If no operand is entered, the command will 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

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.

AUTONAME - Specify Colorization file

Syntax

AUTONAME	AUTO-filename		NONE		*		?
-----------------	----------------------	--	-------------	--	----------	--	----------

Operands

AUTO-filename	The name of the AUTO file to be used with the current Profile
NONE	Requests NO colorization be used, even if a valid AUTO file exists.
*	Requests AUTONAME be reset to the default name (the Profile name)
?	Display the current status of the setting.

Description

Normally, colorization support assumes the name of the AUTO file to use will be equal to the Profile name. i.e. A profile named TXT would default to using an AUTO file named TXT.AUTO. AUTONAME allows you to change this default assumption to whatever AUTO file you choose.

If NONE is entered, any existing AUTONAME value will be removed, and **NO** AUTO file will be used, even if one with the default name exists.

If ? is entered, the current setting will be displayed.

The new value will be saved in the Profile.

AUTOSAVE - Control SAVE Action at END

Syntax

AUTOSAVE	[ON <u>OFF</u> ?]
	[PROMPT <u>NOPROMPT</u>]

Operands

ON | **OFF** The desired AUTOSAVE status

? Display the current status of the setting.

PROMPT |
NOPROMPT The desired Prompt status

Description

If **ON** is specified (or defaulted), SPFLite will 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.

BACKUP - Create a Backup of a file

Syntax

BACKUP		[?]
BACK		
BK		

Operands

? A single ? is taken as a query as to the current number of backups for the current file being edited.

Description

BACKUP is useful to create a Date and Time stamped backup of the file you are currently working on. The command will place the Backup in a **\\$BACKUP** folder in the same folder in which the file resides. If the **\\$BACKUP** folder does not yet exist, it will be created.

BACKUP will also back up the STATE information for the file, if it exists.

Full details of Backup & Restore can be found in [Working with BACKUP & RESTORE](#).

Notes:

Important

BACKUP backs up the most-recently saved version of the file, which will not include any unsaved changes in your Edit session. If there are unsaved changes at the time of the Backup, you will be informed of this condition)

SO ... if you want those changes to be included in the backup **DO A SAVE FIRST!**

Multiple Backups

You may create as many backups of a file as you desire, but SPFLite will only back up a file **ONCE** at a saved level. e.g. if you do a **BACKUP** and immediately do another **BACKUP** without changing / saving the file, the **BACKUP** will fail saying that a current Backup already exists.

As long as the file on disk has been altered, you may create another new backup.

Exception: You can not create a new backup within the same minute as a previous backup, the time-stamp of the backup does not include seconds.

Backup Management

You may choose to manage your Backup files entirely on your own, or you can request SPFLite to assist by specifying various styles of Retention handling. You should review [Working with BACKUP & RESTORE](#) to determine which method suits your requirements the best.

BOM - Turn BOM writing ON / OFF

Syntax

BOM	[ON OFF ?]
------------	-------------------------

Operands

ON OFF	The desired BOM write status
?	Display the current status of the setting.

Description

If no operand is entered, the command wil 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

BOM (Byte Order Marker) prefixes are normally used at the beginning of UTF files to describe which type of UTF encoding is in use.

For UTF8 files, the BOM is optional and in some cases can cause problems when the files are read by some applications. The BOM command, which **only** affects UTF8 files, allows you to control whether SPFLite should write the BOM marker at the beginning of UTF8 files.

If **ON** is specified, the UTF8 BOM sequence will be written preceding the data.

If **OFF** is specified, no BOM will be written.

The value is stored as part of the PROFILE options.

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

?	Display the current status of the setting.
left-col	If the left column boundary is omitted, it is assumed to be 1.
right-col [MAX +]	<p>The right column boundary to be set or the Keyword 'MAX' or + to indicate the maximum record length.</p> <p>You cannot specify the same column for both boundaries. .</p> <p>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".</p> <p>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</p>

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 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.

BROWSE - Open a File for Read-Only

Syntax

BROWSE	[file-name]
	[.Profile-name]
	[%imacro-name %ON %OFF %NONE]
	[/XForm-macro /ON /OFF /NONE]

Operands

file-name The name of the file to be browsed.

The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.

If the file-name does not exist, BROWSE will be cancelled.

.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)

%imacro-name This optional operand allows you to specify an IMACRO (a macro to be run immediately after the file is loaded). If an IMACRO is specified in the file's Profile, this imacro-name will override it.

You can use **%OFF** or **%NONE** to nullify an IMACRO specified in the Profile

You can use **%ON** to activate an IMACRO in the profile which is save in **OFF** mode.

See Profile IMACRO for details.

/XForm-macro This optional operand allows you to specify an XFORM (a macro to allow access to non-standard file types). If an XFORM macro is specified in the file's Profile, this XForm-macro will override it.

You can use **/OFF** or **/NONE** to nullify an XFORM macro specified in the Profile

You can use **/ON** to activate an XFORM macro in the profile which is saved in **OFF** mode.

See Profile XFORM for details.

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.

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 (New)] 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]
---------------	-------------------

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 permanently deleted.

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.

When you issue a **CANCEL DELETE**, you will be presented with a confirmation popup message, asking you to confirm your deletion action. This popup message only 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.

CAPS - Set Keyboard CAPS Mode

Syntax

CAPS [<u>ON</u> OFF AUTO ?]
--

Operands

ON | **OFF** | The desired CAPS status

AUTO

? Display the current status of the setting.

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

Note that when **CAPS ON** is in effect, while you are typing in new characters on a data line, they will be entered in Uppercase regardless of whether you type in lower or upper case. 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 when loaded, and set to 'conform' to the data - **on** if the data is all caps, **off** if mixed case is present. This setting is only temporary. When you open the file again, the file examination is repeated. The "on" or "off" is displayed in *lower case* as 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.

A CAPS AUTO command handling depends on the prior CAPS state

If **CAPS AUTO** is entered, and the current CAPS status is already **AUTO**, then it will cause the command to act as a toggle between **CAPS AUTO:on** and **CAPS AUTO:off**.

CASE - Set Default String Case Handling

Syntax

CASE	[C T ?]
-------------	------------------------------------

Operands

C T	The desired CASE status
?	Display the current status of the setting.

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.

CD - Change Directory

Syntax

CD (File Manager)	<i>directory-name</i>
--------------------------------	-----------------------

Operands

directory-name The desired folder name.

Description

Change Directory. May only be used if File Manager is in FilePath mode. It allows a quick change to be made to the File / Path entry. e.g. **CD D:** or **CD MYFOLDER**.

If the operand is fully qualified (i.e. the 2nd character is a :, the operand replaces File / Folder.

If not fully qualified, the operand is appended to the current File / Folder entry.

If the operand contains spaces, it must be enclosed in quotes.

CHANGE - Change a Data String

Syntax

CHANGE	from-string to-string
	[start-column [end-column]]
	[FIRST LAST <u>NEXT</u> PREV ALL]
	[PREFIX SUFFIX WORD <u>CHAR</u>]
	[C] [Q] [T]
	[LEFT RIGHT]
	[TRUNC]
	[line-control-range]
	[color-selection-criteria]
	[color-change-request]
	[DS CS]
	[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.
C Q T	<p>C - Locate the search string within a defined Comment string. Q - Locate the search string within a defined Quoted literal string. T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).</p> <p>You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.</p> <p>These three operands require a valid Profile with Colorization active.</p>
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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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" .
CS DS	Specifies the Column Shift / Data Shift mode for this particular CHANGE command. See Effect of CHANGE/ALIGN 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.
?	Display the current status of the setting for the default CS / DS option.
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 [<i>clipboard-name</i>]

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.

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.

The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.

.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
- an EFT 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. .

CLS - Clear / Close Debug Window

Syntax

CLS	[T]
-----	-------

Operands

T	Terminate (Close) the DEBUG Window
---	------------------------------------

Description

The SPFLite **CLS** (Clear Screen) command is used to clear or close the DEBUG window used to display MACRO trace and debug output. It allows you to clear all debug output from a prior test before performing another one. The **CLS T** option allows you to completely close the Debug Window, a simple **CLS** command just clears the screen

CLS does **NOT** affect in any way the normal SPFLite Edit window.

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
<i>source-name</i>	Requests use of the codepage called <i>source-name</i> .
?	Display the current status of the setting.

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.

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	[ON OFF ?]
------	------------------

Operands

ON | **OFF** The desired COLS status

? Display the current status of the setting.

Abbreviations and Aliases

COLS can also be spelled as **COL**

Description

If no operand is entered, the command will 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

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
```

COMPRESS - Compress Duplicate Strings

Syntax

COMPRESS	string-1 [string-2] [start-column [end-column]] [line-control-range] [color-selection-criteria] [color-change-request] [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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
color-change-request	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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.

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

<code>COPY</code>	<code>[file-name]</code> <code>[from-line to-line]</code> <code>[{ BEFORE AFTER } line-label]</code>
-------------------	---

Operands

file-name	The name of the file you wish to include in the edit data.
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 } line-label	<p>If specified, the file is copied before, or after, a defined line label.</p> <p>If a before/after line label is not specified, the file is copied into the current edit file in one of the following ways:</p> <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 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 folder searched will be the one from which the currently loaded file was read. 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 (New):

- 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]
	[REP REPLACE]

Operands

line-control-range	<p>The range of lines which are to be included by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification".</p> <p>If no line control range is specified, either via C/CC or M/MM line selection or via the command line, then ALL lines will be selected.</p>
file-name	<p>The name of the file you wish to create. The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.</p>
REP REPLACE	<p>Although you should normally use the REPLACE command when you know the output file already exists, you <u>can</u> use a REP REPLACE operand on CREATE to achieve the same effect. This is simply a convenience item.</p>

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.

See also the REP | REPLACE options above.

To copy the entire Edit data into a new file:

On the Command line, type:
CREATE 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 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** **DCB** 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 (New). CLIP etc.), a pop-up will appear for you to specify the location and file-name desired.

Created with the Personal Edition of HelpNDoc: [Keep Your PDFs Safe from Unauthorized Access with These Security Measures](#)

CSV - Create Filelist CSV file

Syntax

CSV	[ALIGN]
-----	-----------

Operands

ALIGN The optional **ALIGN** operand will cause data for each 'column' to be aligned based on the longest data item.

This will create a CSV (Comma Separated Value) file containing the current displayed Filelist. The optional **ALIGN** operand will cause data for each 'column' to be aligned based on the longest data item. The CSV data is placed in the clipboard and can be easily viewed/edited by using the **CLIP** command

Created with the Personal Edition of HelpNDoc: [Make Help Documentation a Breeze with a Help Authoring Tool](#)

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 50 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 can be used in a normal Edit session, as well as in File Manager. The operands are similar, but there are some differences

CUT (File Manager)	[CB-name] [<u>REPLACE/REP</u> NEW] [<u>APPEND/ADD</u>] [RAW]
CUT (Edit session)	[CB-name] [line-control-range] [ALL] [search-string] (See special Note below) [start-column [end-column]] [color-selection-criteria] [PREFIX SUFFIX WORD <u>CHAR</u>] [C] [Q] [T] [DELETE DEL] [<u>REPLACE/REP</u> NEW] [<u>APPEND/ADD</u>] [RAW]

Operands

CB-name	The name of a Named Private Clipboard. If omitted, the standard Windows clipboard is used. If you use a private name beginning with an Underscore "_", the clipboard will be treated as a temporary clipboard and will be deleted from the CLIP folder after two days. Using temporary clipboard names helps prevent the \CLIP folder from storing Clip data long after it is still useful.
line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
ALL	Note: When optional search criteria are specified, ALL is assumed, but you may optionally code it anyway. ALL is only required when NO operands are specified AND no line-range has been marked.
search-string	A string to further filter what lines you desire to be CUT. You may use any of the available string types (normal, Picture, RegEx etc) that are available in a FIND command. Special Note: the search string, even if a simple string like ABC MUST be enclosed in quotes. e.g. "ABC" This is to prevent it being treated as the

CB-Name operand.

start-col end-col	These column numbers can be used to further restrict where <i>search-string</i> is allowed to occur.
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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-Word characters
<u>CHAR</u>	Locates search-string regardless of what precedes or follows it
C Q T	<p>C - Locate the search string within a defined Comment string.</p> <p>Q - Locate the search string within a defined Quoted literal string.</p> <p>T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).</p> <p>You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.</p>
	These three operands require a valid Profile with Colorization active.
DELETE DEL	Requests that the lines selected for CUT be deleted following the operation.
REPLACE REP	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 .
NEW	If NEW is specified and the CUT is directed to a named clipboard, it requests a verification that the named clipboard does not already exist. If it does exist, you will be prompted for permission to over-write the existing file.
APPEND ADD	The data selected by CUT will be added to the end of any existing data already in the clipboard.

Description

CUT command in File Manager Differences

The **CUT** command in File Manager has many similarities to the Edit mode **CUT** command, but also some significant differences.

- The **Line-Control range**, **ALL**, **X/NX**, **U/NU** and **DELETE** options are not supported.
- The data copied to the clipboard is the full file path, enclosed in Double-Quotes. (**Not** the contents of the file)
- Selected lines must be specified via **C** / **CC** / **Cnn** line commands, there are no selection criteria available as operands to the CUT command

In an **EDIT** session, **CUT** saves copies of the requested 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](#).

COPY vs CUT

Windows has used **Copy/Cut** for years, and we are all familiar with their **Windows** meaning.

The SPFLite **CUT** command name is however a hold-over from the original support in ISPF, We are 'stuck' with the name and the way it functions..

In SPFLite the **CUT** command can act like either of the Windows COPY/CUT commands. **How?** It depends on how you select the lines.

COPY Mode:

The lines are selected via **C/CC** line commands, or text selection criteria, **AND** you **do not** specify the **DELETE** operand.

CUT Mode:

The lines are selected via:

M/MM line commands

C/CC line commands, or text selection criteria, **AND** you specify the **DELETE** operand.

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 putting data into the clipboard.

Specifying the Data to CUT

Two methods of specifying the line range of the selected 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 [M/MM](#) 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 CUT command also provides for selection based on the contents of the data line. This search ability is similar to what is provided by FIND/CHANGE. It can be used in conjunction with a specified line range, or if no line-range is specified it assumes all lines are to be searched.

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.

DCB - Set File Characteristics

Special Note:

Because the **EOL**, **LRECL**, and **RECFM** settings are logically intertwined in controlling how your files are read and written, validating and changing these critical values needs to be done together. All three of these commands are handled by a single command processor to ensure that no illogical combinations of operands are created.

This means that the **DCB** command can be used to alter any of these critical values, either singly, or in combinations.

When one or more of the 3 categories (RECFM, LRECL EOL) are omitted the existing values are swapped in for the purposes of validation.

e.g. if **DCB EOL CRLF** is entered, the existing values for **LRECL** and **RECFM** will be assumed for the validation process.

Syntax

DCB	?	
		[RECFM xxx] [LRECL nnn] [EOL yyy]
Where:	?	Display the current settings.
	RECFM xxx	XXX may be: U The file uses Undefined type records. i.e. The records are separated by unique delimiters which should be specified using the EOL operand. 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 operand. As well, the length of the fixed records should be set via the LRECL operand. V The file uses Variable record format. Typically this means it 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.

LRECL nnn

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: **RECFM=U, LRECL=0,EOL=CRLF**

For SPFLite, LRECL generally means the **fixed** logical record length. Specifying record-length = **0** does not mean the value is zero, but rather, that there **is no** (arbitrary) length.

Files originating on a mainframe or a nonstandard system may contain a fixed-length record format. This parameter allows you to specify a given fixed length for a record.

Fixed-length records may be stored either with or without *End-of-Line* (**EOL**) characters. The value 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** as **NONE**. File types declared with NONE must have a defined *record-length* value that is not **0** (zero). You would also have *record-format* = **F** to signify a fixed-length file, or *record-format* = **V** to signify a variable-length file that contains Record Descriptor Words (RDW's) instead of EOL delimiters.

EOL yyy

The following are supported for yyy:

NONE There are no line separator characters used. The file must have an **LRECL** greater than 0 which will be used to perform the line separator function.

CRLF Use the carriage return / Line Feed pair 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 NewLine character as the line separator.
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

Description

These three settings control the record deblocking to be used when reading/writing the file. The values are stored in the file Profile and are used when reading and writing the files.

For additional information on these values and their effect, see ["Handling Non-Windows Text Files"](#).

Handling of End-Of-Line AUTONL

Note: **AUTONL** is used to allow for lone CR characters in a text file.

Lone CR characters will **always** be treated as a new line request,

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 EOL 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.

Both a lone **LF (line feed)** or a lone **CR (carriage return)** will be 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.

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 EOL AUTO 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.

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>]] [C] [Q] [T] [start-column [end-column]] [line-control-range] [color-selection-criteria] [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: If you want to use **DUP** or **NOTE** as normal search strings they must be enclosed in quotes.

DUP Specifies this command is to delete adjacent duplicate lines from the line range. 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
xNOTE
ZNOTE **NOTE** specifies this command is to delete all **=NOTE>** lines from the line range.

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 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.

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. The default for string searches is **NEXT**. 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.

C
Q
T

C - Locate the search string **within** a defined Comment string.
Q - Locate the search string **within** a defined Quoted literal string.
T - Locate the search string **within** plain text (i.e. Not in a Comment or Quoted string).
You may enter more than 1 of **C Q** or **T** to customize the selection. They are tested in an **OR** relationship.

These three operands require a valid Profile with Colorization active.

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. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

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-

A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-

criteria	selection-criteria are discussed in "Color Selection Criteria Specification" .
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:

- 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.

Repeatable DELETE

Just as **FIND** and **CHANGE** can be selectively repeated using **RFIND** (or **RLOC****FIND**) 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, **RLOC****FIND** 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 **RLOC****FIND**) 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.

DIFF - Compare two files

Syntax

```
DIFF          no operands at all

              or

LIST

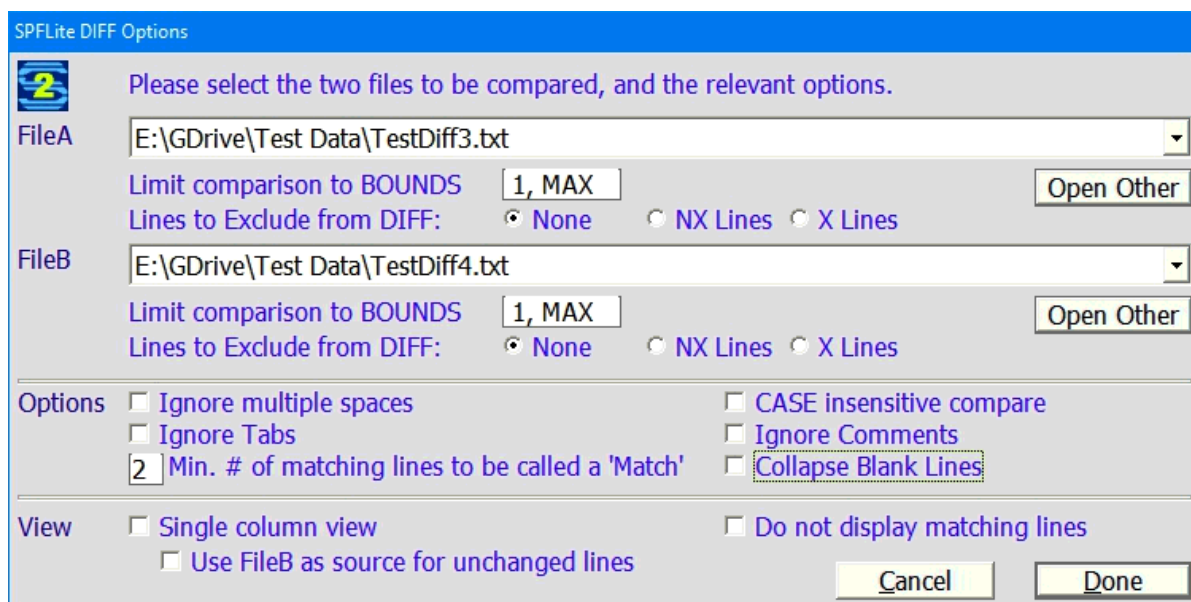
              or

{ FileA-TabNum | FileA-filename } }
{ FileB-TabNum | FileB-filename } }
```

Further details on using DIFF, including sample reports, can be found in ["Working with DIFF"](#).

Entered with No Operands

If DIFF is entered without any operands, you will be presented with the following pop-up dialog to specify your desired parameters.



The image shows a dialog box titled "SPFLite DIFF Options". It contains fields for "FileA" and "FileB", each with a text input and a dropdown arrow. Below each file field are options for "Limit comparison to BOUNDS" (a text input with "1, MAX") and "Lines to Exclude from DIFF:" (radio buttons for "None", "NX Lines", and "X Lines"). There are "Open Other" buttons next to the bounds input. The "Options" section has checkboxes for "Ignore multiple spaces", "Ignore Tabs", "CASE insensitive compare", "Ignore Comments", and "Collapse Blank Lines". A text input for "Min. # of matching lines to be called a 'Match'" is set to "2". The "View" section has checkboxes for "Single column view", "Do not display matching lines", and "Use FileB as source for unchanged lines". "Cancel" and "Done" buttons are at the bottom right.

SPFLite DIFF Options

Please select the two files to be compared, and the relevant options.

FileA E:\GDrive\Test Data\TestDiff3.txt

Limit comparison to BOUNDS 1, MAX

Lines to Exclude from DIFF: ☒ None ☐ NX Lines ☐ X Lines

Open Other

FileB E:\GDrive\Test Data\TestDiff4.txt

Limit comparison to BOUNDS 1, MAX

Lines to Exclude from DIFF: ☒ None ☐ NX Lines ☐ X Lines

Open Other

Options ☐ Ignore multiple spaces ☐ CASE insensitive compare

☐ Ignore Tabs ☐ Ignore Comments

2 Min. # of matching lines to be called a 'Match' ☐ Collapse Blank Lines

View ☐ Single column view ☐ Do not display matching lines

☐ Use FileB as source for unchanged lines

Cancel Done

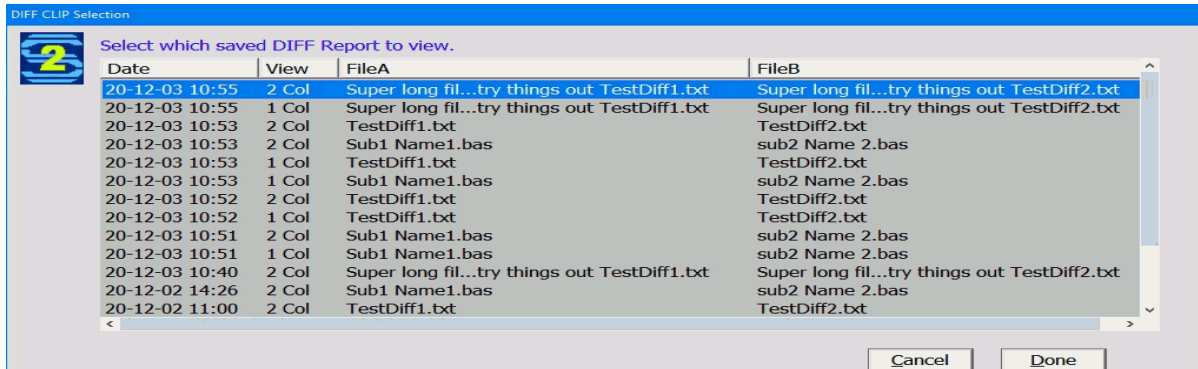
The FileA and FileB combo boxes will be pre-populated with the list of files currently open in an SPFLite tab.

If you wish to select one of these, simply pull down the list using the right-hand arrow and select the file.

If you want to select some other file not currently open, click the "Open Other" button to the right and a normal file selection dialog will be presented for you to choose the file. When "Other" files are selected, they will be opened in a new tab automatically, prior to performing the compare.

Entered with simply - DIFF LIST

If entered as simply **DIFF LIST**, SPFLite will display a popup allowing you to choose from a list of available saved DIFF reports. The selected report will be loaded for review in a normal DIFF CLIP session. The pop-up will look like



Entered with detailed Operands

FileX-TabNum

You may enter a .simple number to indicate the Tab # currently containing the desired file. The File Manager tab is 1, the next tab to the right is 2 etc. Again Tab 1 (FM) is not allowed.

FileX-FileName

You may enter the full name of the desired file. This can be a file currently open in one of the tabs, or some other file not currently Open. If a file is not currently Open, it will be loaded into a new tab prior to the DIFF compare.

When entered with operands, the FileA and FileB input will be populated with those files, and the Pop-Up dialog will appear to allow you to Enter/Alter the other DIFF options. Press **Done** to continue.

Operand Retention

The optional settings for DIFF will be saved and re-established for each DIFF invocation. One exception is the **Bounds** values. As these are normally quite volatile, they will only be saved for the current day, and will be reset to **1, MAX** thereafter.

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.

DO - Invoke a KB Command file

Syntax

DO	<code>command-file</code>
----	---------------------------

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.

EDIT - Open a File for Editing

Syntax

EDIT	[file-name]
	[.Profile-name]
	[%imacro-name %ON %OFF %NONE]
	[/XForm-macro /ON /OFF /NONE]

Operands

file-name	<p>The name of the file to be editedEdit.</p> <p>The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.</p> <p>If the file does not exist, it will be created as an Empty file. If the file is not really desired, issue a CANCEL DELETE command to exit and delete the empty file.</p>
.Profile-name	<p>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)</p>
%imacro-name	<p>This optional operand allows you to specify an IMACRO (a macro to be run immediately after the file is loaded). If an IMACRO is specified in the file's Profile, this imacro-name will override it.</p> <p>You can use %OFF or %NONE to nullify an IMACRO specified in the Profile</p> <p>You can use /ON to activate an IMACRO in the profile which is save in OFF mode.</p> <p>See Profile IMACRO for details.</p>
/XForm-macro	<p>This optional operand allows you to specify an XFORM (a macro to allow access to non-standard file types). If an XFORM macro is specified in the file's Profile, this XForm-macro will override it.</p> <p>You can use /OFF or /NONE to nullify an XFORM macro specified in the Profile</p> <p>You can use /ON to activate an XFORM macro in the profile which is saved in OFF mode.</p> <p>See Profile XFORM for details.</p>

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.

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 (New)), 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.

EFT - Extended File Types

Syntax

EFT [TEST <i>sample-file-name</i>]
--

Operands

TEST To request evaluation of the *sample-file-name* against the currently displayed EFT set of rules.

sample-file-name A full filename (Drive/Path/Filename/Extension)

Description

With no operands, the **EFT** command will open a special edit session containing the current list of **EFT** (Extended File Type) rules.

You can then edit, add or delete as you need. Or re-order the statements to achieve your desired result if you have a complicated set of directives.

When complete, simply **END** the session. The new directives will take immediate effect.

Or enter **CANCEL** to leave without making changes.

TEST Mode

EFT TEST mode, which is only allowed while in (EFT Edit) mode, allows you to 'trial run' a modified set of EFT rules without saving them and performing a real file OPEN to see the effect.

EFT TEST will perform a match of the *sample-file-name* against each rule in the current display. If the *sample-file-name* would be matched by the rule, that rule will be hi-lighted. The following screen shots show this effect.

Here's the current (EFT Edit) display and the EFT TEST command:

```

EFT-Edit - SPFLite(v2.7.22328)
File Manager (EFT Edit)
Command > left test D:\FOLDER\FILEABC.INC
Scroll > CSR

=COLS> -----1-----2-----3-----4-----5-----6-----7-----+
***** Top of Data *****
000001 ; Enter your EFT entries following this line
000002 TXTMNO1 = TXT
000003 "G:\*.TXT" = BAS
000004 "CFGMaint EXP*.TXT" = BAS
000005 "*ABC.INC" = ABCBAS
000006 ; The following items were Auto-Converted from old Profile USING entries
000007 CPP = C
000008 H = BAS
000009 FLATFILE = BAS
000010 INC = BAS
***** Bottom of Data *****

```

EFT Edit Lines: 10 Cols 1 to 75 Bnds: MAX INS T DS 5

Here's the result of the command:

```

EFT-Edit - SPFLite(v2.7.22328)
File Manager (EFT Edit)
Command >
Scroll > CSR

2 EFT entries matched the filename and were marked

=COLS> -----1-----2-----3-----4-----5-----6-----7-----+
***** Top of Data *****
000001 ; Enter your EFT entries following this line
000002 TXTMNO1 = TXT
000003 "G:\*.TXT" = BAS
000004 "CFGMaint EXP*.TXT" = BAS
000005 "*ABC.INC" = ABCBAS
000006 ; The following items were Auto-Converted from old Profile USING entries
000007 CPP = C
000008 H = BAS
000009 FLATFILE = BAS
000010 INC = BAS
***** Bottom of Data *****

```

EFT Edit Lines: 10 Cols 1 to 75 Bnds: MAX INS T DS 5

Note that in this case there are two rules which match the sample filename.

Whenever there is more than 1 rule matching the sample filename it is important to ensure the order of these rules in the list is correct. In the example above, if the **INC=BAS** had been first, then **no matching filename** would ever be selected by the **"*ABC.INC" = ABCBAS** rule since the simpler rule would have been encountered, and used, first.

Once you are happy that your changes to the EFT rule set are working properly, then **END** the session to save the EFT rule set.

For full information of **EFT** usage and **EFT** syntax, see ["Extended File Types"](#).

EMACRO - Set pre-Save Macro

Syntax

```
EMACRO      [ NONE ]  
            | Macro-name ]  
            [ ? ]  
            [ ON | OFF ] ]
```

Operands

NONE Will nullify any existing EMACRO string.

? Display the current status of the setting.

Macro-name The Macro command you wish invoked.

If **ON** or **OFF** are specified, they indicate:

ON - The macro is activated and will be used on each file save.

OFF - The macro name is saved, but is "dormant", it will not be used on file save.

Description

The **EMACRO** command allows you to specify a Macro which will be run immediately before an SAVE of the file.

NONE If **EMACRO** is issued with no operands, it will simply display the current **EMACRO** setting. Therefore, in order to nullify an existing **EMACRO** **NONE** is used to represent the "" condition.

The effect of **EMACRO** will be seen the next time a file using this profile is saved.

Actions when only some operands are entered

- When only Macro-name is entered
 - If no current Macro-name, **ON** is assumed
 - If there **is** a current Macro-name, the existing **ON/OFF** state is assumed.
- When only **ON/OFF** is entered
 - If no current Macro-name, an error message is displayed
 - If there **is** a current Macro-name, it is kept and the new **ON/OFF** setting saved.
- When **NONE** is entered, any other operands are ignored, and any existing Macro-name is nullified.

END - End the Edit Session

Syntax

For the **File Manager** Tab

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.

For all normal **Edit/Browse/View** Tabs

END

Operands

None

Description

The **END** command may be used in an Edit or Browse session, or Set Edit, EFT-Edit, 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 **(New)**, 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.

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 should be specified as a decimal value. The value of number cannot be zero.
?	Display the current status of the setting.

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.

As an Edit primary command, **ENUMWITH** cannot be issued from the File Manager.

EOL - Set End-of-Line delimiter

Syntax

Note: The **EOL** command has been deprecated, and although it still works as before, you should begin to use the new [DCB](#) command, which replaces EOL.

EOL	[CRLF CR LF NL AUTO AUTONL hh hhh NONE ?]
------------	--

Operands

The full description of the **EOL** operands will now be found in the [DCB](#) command.

EXCLUDE - Exclude Lines Edit Mode

Syntax

EXCLUDE	[search-string]
(Edit mode)	[start-column [end-column]]
	[FIRST LAST NEXT PREV ALL]
	[PREFIX SUFFIX WORD CHAR]
	[C] [Q] [T]
	[line-control-range]
	[color-selection-criteria]
	[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.
C	C - Locate the search string within a defined Comment string.
Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string. You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.
	These three operands require a valid Profile with Colorization active.
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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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.xx

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"**

For more details on mask options see ["Extended File Pattern Support"](#)

EXIT - Terminate SPFLite Session

Syntax

<code>EXIT =X</code>	<code>[NOREOPEN]</code>
	<code>[<u>END</u> CANCEL [DELETE]</code>

Operands

NOREOPEN	<p>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.</p> <p>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.)</p>
<u>END</u>	<p>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.</p>
CANCEL	<p>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.</p>
DELETE	<p>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.</p>

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**. This supports the old IBM 'quick exit' function.

FAVORITE - Add Current File To A Favorite List

Syntax

FAVORITE [<u>Favorite</u> 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.

FF - Find In Files

Syntax

FF	string [start-column [end-column]] [CHARS WORD PREFIX SUFFIX] [NF]
-----------	--

Operands

string	<p>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.</p> <p>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.</p>
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	<p>Specifies the 'search context' for the string, the same as is done for the FIND command in the editor.</p> <p>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.</p>
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**

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 bring 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	search-string	(Perform a
	normal FIND)	
Alias - FF	[start-column [end-column]]	
	[FIRST LAST NEXT PREV ALL]	
	[PREFIX SUFFIX WORD CHAR]	
	[C] [Q] [T]	
	[LEFT RIGHT]	
	[line-control-range]	
	[color-selection-criteria]	
	[color-change-request]	
	[MX DX]	
	[TOP]	

Operands

Operand descriptions are for 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.
C	C - Locate the search string within a defined Comment string.
Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).
	You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.

	These three operands require a valid Profile with Colorization active.
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	<p>The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification".</p> <p>See also the Line Range / Pending Copy Conflict discussion below.</p>
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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" .
MX	<p>MX requests that all lines which DO contain search-string be excluded from the display following command processing.</p> <p>MX = Make Excluded.</p>
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

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

Difference between FF and FIND

The **FF** alias for the **FIND** command is identical to **FIND** with the following distinction:

- **FF** will honor a pending **CC/CC** block as the line range to be searched.
- **FIND** will ignore the pending **CC/CC** block so that you can use **FIND** to search for a position where you might want to place an **A** or **B** command.

Normal Usage

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.

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

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 ]  
          [ C ] [ Q ] [ T ]  
          [ line-control-range ]  
          [ 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.
C	C - Locate the search string within a defined Comment string. Q - Locate the search string within a defined Quoted literal string. T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string). You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship. These three operands require a valid Profile with Colorization active.
Q	
T	
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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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.

GLUEWITH - Specify Join String for GLUE operations

Syntax

GLUEWITH [<i>string</i> ?]

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 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.

? Display the current status of the setting.

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 CFG configuration 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.

See [G / GG - Glue Lines Together](#) and [TG / TGG - Text Glue Lines](#) for examples of using **GLUEWITH**.

HELP - Display Online Help

Syntax

HELP [ALL] [search-topic(s)]
--

Operands

search-topic(s) One or more (up to 8) search terms.

Description

If **no** operands are provided, HELP will either begin in ["Working with SPFLite"](#) (if you are in Edit) or ["Working with File Manager"](#) (if you are in FM)

The command operates slightly differently depending on how many search topics you enter.

A Single Operand

The following single operands are supported:

MACRO or MACROS	This will open Help at the Introduction to Macros section
THINBASIC or BASIC	This will open the thinBasic support Help file
*	This will open the Main SPFLite Help file at the Introduction
CMD or CMDS or PRIM or PRIMARY	This will open the Main SPFLite Help file at the List of Primary commands. The operation is sensitive to the current tab type. If you are in File Manager, it will open at the File Manager Primary command list. If you are in Edit, it will open at the Edit Primary command list.
LINE or LINES	This will open the Main SPFLite Help file at the List of Line commands. As with the previous item, the operation is sensitive to the current tab type. If you are in File Manager, it will open at the File Manager Line command list. If you are in Edit, it will open at the Edit Line command list.
Any Primary, Line command or Macro Function Name	For example, to learn how to use the SORT command, issue HELP SORT . To learn about Text Flow, enter HELP TF . For details on a Macro function enter HELP GET_ARG\$ (for example)

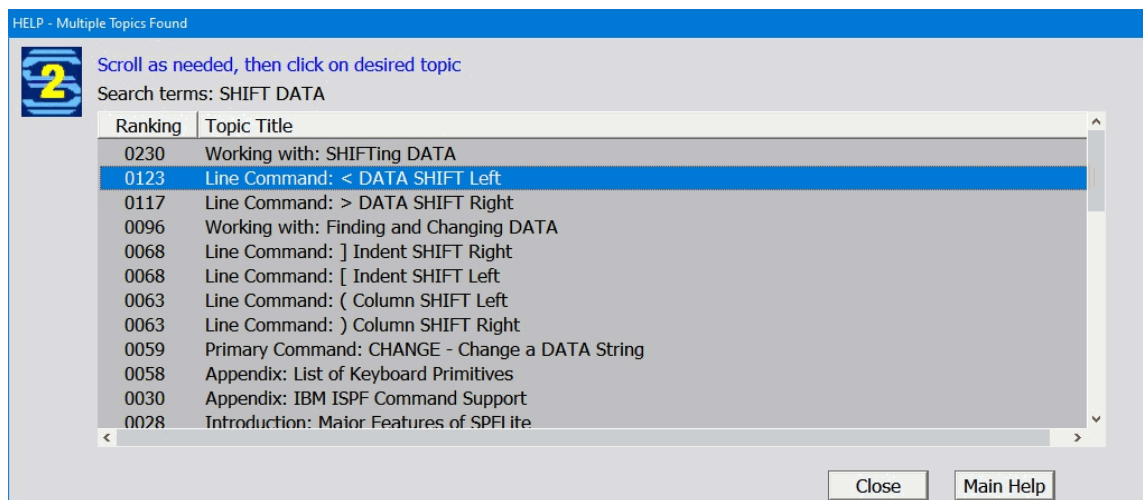
Multiple Operands

When multiple operands are entered, HELP performs a more exhaustive search based on an extract of all important keywords in the full Help file. The search treats all the operands as being in an AND relationship. i.e. find a topic that contains operand-1 AND operand-2 AND operand-3 etc.

Note: If you want to perform an exhaustive search on a single word, simply insert **ALL** as the first operand.

If only 1 topic matches the search operands, it will be directly opened.

More typically, the search will return multiple possibilities. When it does, you will be presented with the following pop-up:



In the example above, the command entered was **HELP SHIFT DATA**. The search found multiple related topics. The list is ordered with the most relevant items first, and any search terms in the topic titles are upper-cased to make them stand out. The left column displays the internal ranking. This number is a measure of how many 'hits' each topic has. The 'hits' are also weighted based on where they occur. e.g. In the Topic Title, in the detailed text, etc.

At this point, simply click on your preference, and the Help file will be opened to that specific Topic.

The pop-up will remain open. If you are not happy with your choice, simply click a different topic from the list. The previous topic will be closed and your new selection will appear.

When you feel the correct topic has been found, click the **Close** button to dismiss the pop-up.

You can also select the **Main Help** button to simply open the Help file at the Introduction.

HEX - Enable Hex Display of Data

Syntax

HEX	[ON OFF ?]
------------	---------------------------------------

Operands

ON OFF	The desired hex display status
?	Display the current status of the setting.

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.

If no operand is entered, the command will 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

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 or FILE Lines

Syntax

HIDE	[<u>X</u> FILE]
	[<u>ON</u> OFF]
	[?]

Operands

<u>X</u> FILE	X indicates you want to turn ON/OFF the display of the Excluded Line Marker FILE indicates you want to turn ON/OFF the display of the Multi-Edit =FILE> Markers
<u>ON</u> OFF	The desired Hide status
?	Display the current status of the setting.

Description

HIDE X

The **HIDE X** command controls the display of the "excluded-line placeholder". Excluded lines are normally collapsed to this placeholder line indicating how many lines are represented. In **HIDE X ON** mode, the excluded-line placeholder is removed from the display. The data lines themselves are still present in the edit file.

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 X OFF** command, or else you must use the **ALL** option on **FIND**, **CHANGE** and similar commands.

See [Working with Excluded Lines](#) for more information.

HIDE FILE

The **HIDE FILE** command controls the display of the File Marker lines in [Multi-Edit mode](#). These lines are inserted at the beginning of each file's data lines in a Multi-Edit session.

But when numerous files are loaded there are times these lines can be a distraction. Entering **HIDE FILE OFF** will remove them from the display. When they are removed there is **no** indication at all of their position. To See them again simply enter **HIDE FILE OFF**.

Defaults

If neither **X** nor **FILE** are entered (i.e. just ON or OFF) the On/Off setting will be applied to the **X** setting. To control FILE you must enter the **FILE** operand.

If you enter X or FILE, but no On/OFF operand, **ON** will be assumed.

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.
?	Display the current status of the setting.

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.

IMACRO - Set Initial Macro

Syntax

```
IMACRO      [ NONE ]  
            | Macro-name ]  
            [ ON | OFF | ? ] ]
```

Operands

NONE Will nullify any existing IMACRO string.

? Display the current status of the setting.

Macro-name The Macro command you wish invoked.

If **ON** or **OFF** are specified, they indicate:

ON - The macro is activated and will be used on each file load.

OFF - The macro name is saved, but is "dormant", it will not be used on file load.

Note: the ON/OFF state can be over-ridden at file open time by adding a **/ON** or **/OFF** to the [EDIT](#), [BROWSE](#) or [VIEW](#) command. In File Manager, the **/ON** or **/OFF** can be entered as a line command.

Description

The **IMACRO** command allows you to specify a Macro which will be run immediately after loading a file.

NONE If **IMACRO** is issued with no operands, it will simply display the current **IMACRO** setting. Therefore, in order to nullify an existing **IMACRO**, **NONE** is used to represent the "" condition.

The effect of **IMACRO** will only be seen the next time a file using this profile is loaded.

Actions when only some operands are entered

- When only Macro-name is entered
 - If no current Macro-name, **ON** is assumed
 - If there **is** a current Macro-name, the existing **ON/OFF** state is assumed.
- When only **ON/OFF** is entered
 - If no current Macro-name, an error message is displayed
 - If there **is** a current Macro-name, it is kept and the new **ON/OFF** setting saved.
- When **NONE** is entered, any other operands are ignored, and any existing Macro-name is nullified.

INSTANCE - Switch to another Instance

Syntax

```
INSTANCE      Instance-Name  
INST          [ END | CANCEL | CAN | KEEP      ]  
              [ -F [ FileOpen-FileName | * ] ]  
              [ -D DO-macroName                ]
```

Operands

<i>Instance-Name</i>	The name of the Instance to switch to. It IS legal to 'switch' to the current running Instance. If the instance is not running it will be opened. If it has never been created, the normal Instance creation popup will also be displayed.
END	If there are Edit tabs in existence, they will be closed via an END command and the SPFLite session terminated. The END command will utilize whatever AUTOSAVE options are in effect. The Most Recent Files list (MRF List) will be updated so that the next restart will re-open the same files.
CANCEL CAN	If there are other Edit tabs in existence, they will be closed via a CANCEL command, and the session terminated. The MRF list will be nulled, so no files will be automatically re-opened on the next restart.
KEEP	All current tabs will be untouched, and the current session will remain open. All that will happen is the startup of the named <i>Instance-Name</i> . (using any -F or -D options)
-F	This can specify a filename or a single *. The * requests use of the standard _FILEOPEN.TXT file. You may specify any other filename you choose. The format of the file is given in FileOpen Startup Files . If you specify an unqualified filename, the folder searched will be the SPFLite HomeFolder (the same folder as the normal _FILEOPEN.TXT file).
-D	Specifies the name of a DO macro to be invoked on startup of the new session. This option cannot be used if the specified <i>Instance-Name</i> is equal to the current session.

Description

CAUTION: The INSTANCE command is intended for advanced users of SPFLite who are familiar with the use of Instances. If you are not familiar with Instances, please read [Instances and Configurations](#) before attempting use of this command.

The **INSTANCE** command allows you to request a switch to another SPFLite Instance.

When Edit tabs are still open, you may add the **END**, **CANCEL** or **KEEP** operands to describe how they are to be handled. See **Operands** above.

The INSTANCE command, to support flexibility in usage, performs several actions to assist:

- Except when the *Instance-name* equals the current Instance Name, it will create a file _FILEOPEN.TXT in the SPFLite Home folder (the one containing the CFG file) which contains the filenames of all open tabs. This file is in the format used for the Command line -FILEOPEN operand. You can reference this list in the newly opened Instance by coding the **-F *** operand.
- INSTANCE can 'point' at the current running Instance. See [Instances and Configuration](#) for how this might be useful.
- The command will perform preliminary validation of FileOpen and DO macro requests for correctness before attempting the Instance switch.

JOIN - Join lines Using Find/Change Strings

Contents of Article

[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 ?](#)

[Line joining and line exclusion](#)

Syntax

```
JOIN      P'from-string' | R'from-string'  
          [ to-string ]  
          [ FIRST | LAST | NEXT | PREV | ALL ]  
          [ PREFIX | SUFFIX | WORD | CHAR ]  
          [ C ] [ Q ] [ T ]  
          [ line-control-range ]  
          [ color-selection-criteria ]  
          [ 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 <i>change string</i> 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.
<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.

C	C - Locate the search string within a defined Comment string.
Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).
	You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.
	These three operands require a valid Profile with Colorization active.
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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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 **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 of 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-

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.

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.

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.

NFIND P'='

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.

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.

KEYMAP - Display Keyboard Settings Dialog

Syntax

KEYMAP [LIST QUERY string]
--

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. |
| QUERY | Requests a search of currently assigned key definitions for the specified string. This can be used when you 'forget' just where you may have assigned a particular KB command string. |

Abbreviations and Aliases

KEYMAP can also be spelled as **KEYS**, **KEY** or **KBD**, **LIST** can be just **L**, **QUERY** can be just **Q**.

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](#)".

Full descriptions of available primitive keyboard functions can be found in the Appendix sections: [Introduction to Keyboard Primitives](#), [Index to Keyboard Primitives](#) and [List of Keyboard Primitives](#).

Using KEYMAP LIST and QUERY results

In order to help you examine the contents of your **LIST** or **QUERY request**, SPFLite will:

- Collect the information for your request
- Format it into a text lines
- Store the text lines in the Windows clipboard
- A CLIP Edit window is opened in SPFLite to allow you to examine the contents of your 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.

LABEL - Manipulate Line Labels

Syntax

The LABEL command comes in four basic syntax forms	
<u>The Delete Form</u> LABEL	Delete a label from a line <i>.Label</i> or <i>.LineNum</i>
<u>The Search Form</u> LABEL	Add a label to a 'searched for' line <i>.Label</i> search-string [+ -nnn] Note: No space between + - and nnn [start-column [end-column]] [FIRST LAST <u>NEXT</u> PREV] [PREFIX SUFFIX WORD <u>CHAR</u>] [C] [Q] [T] [line-control-range] [color-selection-criteria]
<u>The Re-Assign Form</u> LABEL	Assign / Reassign a Label <i>.Label</i> <i>'Current-Label</i> [+ -nnn] Note: No space between + - and nnn
<u>The Search Tag Form</u> LABEL	Add a label to a 'searched for' line with a :Tag assigned <i>.Label</i> : <i>TagName</i> [<u>NEXT</u> PREV FIRST LAST] [line-control-range] [+ -nnn] Note: No space between + - and nnn

Operands

.Label

The line label to be manipulated:

The Delete Form: The label to be deleted.

The Search Form: The label to be assigned.

The Re-Assign Form: The label to be moved or assigned.

The Search Tag Form: The label to be assigned.

In assigning a Label, if an existing label is present, it will be replaced. If the label exists on a different line, it will be removed from that other line.

<u>.Current-Label</u>	An existing assigned label to be used as the new assignment line.
search-string	The search string that identifies the line 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 value.
LAST	Starts at the bottom of the data and searches backward to find the last occurrence of search value.
<u>NEXT</u>	Starts at the first position after the current cursor location and searches ahead to find the next occurrence of search value. NEXT is the default.
PREV	Starts at the current cursor location and searches backward to find the previous occurrence of search value.
C	<p>C - Locate the search string within a defined Comment string.</p> <p>Q - Locate the search string within a defined Quoted literal string.</p> <p>T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).</p> <p>You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.</p>
Q	
T	
PREFIX	<p>The three above operands require a valid Profile with Colorization active.</p> <p>Locates search-string at the beginning of a word.</p>
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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-	A request for selection based on the highlight color of the search-string. The

selection-criteria full syntax and allowable operands which make up a color-selection-criteria are discussed in ["Color Selection Criteria Specification"](#).

+ | -nnn This specifies an 'adjustment value' for the assignment forms of the command. It allows you to 'adjust' the line number found by the search routine either forward (+) or backward(-) from the located line number.
Note1: The adjustment will NOT adjust past the 1st or last line, nor will an attempt be treated as an error.
Note2: There must be no space between the

Abbreviations and Aliases

LABEL can also be spelled as **LBL** or **LAB**
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

This command allows you to perform various manipulations of individual Line Labels. You can Add, Delete or re-assign labels. Here are a few commented examples.

LABEL .AA "First" 10 20 FIRST	Assign the label .AA to the first line containing the string "First" between columns 10 and 20.
LABEL .DEAD	Remove the label .DEAD wherever it is located
LAB .ZFIND	Remove ANY label from the last line found by a FIND command.
LABEL .AA .AA +3	Re-assign the label .AA to 3 lines past its current location.
LBL .R :BX -1 LAST .10 .200	Assign label; .R to 1 line prior to the Last occurrence of the tag :BX in lines 10 thru 200.
LAB .XX fred PREFIX BLUE .AA .BB	Assign .XX to the next line containing the Prefix fred , in color BLUE , between lines marked by the .AA and .BB line labels.

Created with the Personal Edition of HelpNDoc: [Step-by-Step Guide: How to Turn Your Word Document into an eBook](#)

LC - Lower Case a Range of Lines

Syntax

LC	[line-control-range] [ALL] [MX DX]
-----------	--

Operands

line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
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] [<u>FIRST</u> LAST ALL] [TOP]
-------------	---

Operands

line-command Any standard SPFLite Edit line command. The command may be quoted or unquoted, and must be between 1 and 8 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 itself contains spaces, or might be confused as an actual **LINE** operand, it 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. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)".

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.

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**, **RLOC****FIND** 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 an command that overlaps the syntax of the **LINE** primary command, then the **line-command** operand itself 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

File Manager:	
LOCATE	<i>name</i> 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.
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 DIFF ERROR EXCLUDED FILE FIND HANDLE KEEP LABEL LONG [n] NOTE/xNOTE/ZNOTE NU/U PAGE [n] SPECIAL SIZE n TAG tagname } [NEXT FIRST LAST PREV CURRENT ALL [MX]] [colorname STD] [TOP]

Operands

line-label	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-num	An optional range of lines to be searched when using the generic format LOCATE command.
line-label-	
a line-	
num-a	This will negate or reverse the generic search request to locate the next line
line-label-	
b line-	
num-b	
NOT	

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> .
DIFF	DIFF is simply an alias for U , the DIFF report marks lines which are altered as USER lines, so a LOCATE DIFF command takes you to the next marked DIFF block.
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.
HANDLE	Locate the next line containing a Handle.
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 <u>of any kind</u> , 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.
NEXT 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.
<i>colorname</i>	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 use 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. 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

LOOPCHECK - Turn loop detection ON or OFF

Syntax

<code>LOOPCHECK</code> <code>[ON OFF ?]</code>
--

Operands

`ON` | `OFF` The desired LOOPCHECK status

`?` Display the current status of the setting.

Description

The normal state for SPFLite processing is to always activate loop detection. Note: it can be suppressed at startup time via the command line **_NOLOOP** operand. Details of how loops are handled can be found in [Crash Handling](#).

LOOPCHECK allows you to 'turn on' and 'turn off' this detection **during** a session.

LRECL - Set record Length

Syntax

Note: The **LRECL** command has been deprecated, and although it still works as before, you should begin to use the new [DCB](#) command, which replaces LRECL.

LRECL [n ?]

Full details of the **LRECL** command operands will now be found in the [DCB](#) command.

MACLIB - Specify Unique Macro Folder

Syntax

MACLIB [<i>Macro-SET-Name</i> NONE]

Operands

Macro-SET-Name The name of a [SET](#) symbol containing the filename string, or **NONE** to remove any existing value(s)

Description

The MACLIB command will setup an alternate Macro folder to be used in place of the normal \Documents\MACROS\ folder.

The specified folder(s) will be used when editing files using the current Profile.

The MACLIB support also allows you to specify multiple folders as search candidates for the macro. And because the MACLIB value can thus be quite long, it would be very difficult to enter the value correctly as an operand on the command line. Therefore the **MACLIB** command expects only the name of a [SET](#) symbol. The full SET symbol name is **MACLIB.name**

To enter the needed SET variable, do the following:

1. Enter the primary command **SET**. SPFLite will open a (SETEdit) session containing any current SET variables.
2. Add a new line containing, for example:
MACLIB.MYNAME = C:\MyStuff\MyMacros\|C:\Users\Me\Documents\SPFLite\MACROS\
3. Enter the primary command **END**. The SET variables will be saved

The MYNAME above is your choice, it must match the *Macro-SET-Name* you entered on the **MACLIB** command. The example above adds the normal MACROS folder after the new user folder, but there is no requirement to do so. You may enter any folder(s) you desire.

When entering multiple MACLIB paths, separate them with a "|" character.

Dynamic Maclib Specification

Creating the SET entry as described above is mandatory. If you prefer to NOT use MACLIB to set a Profile to switch libraries, you can dynamically request a different maclib by prefixing the macro name with the SET name.

e.g. AA:*macname* where AA is the MACLIB.AA entry in the SET table, and *macname* is the macro name.

Note: If MACLIB is specified in the Profile, and the dynamic prefix is also used, the dynamic specification takes precedence.

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. If possible, the current Pattern Mask will be added as a filter to each created entry.

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	[ON OFF ?]
-------------	---------------------------------------

Operands

ON OFF	The desired MARK status
?	Display the current status of the setting.

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.

If no operand is entered, the command wil 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

The **ON/OFF** value of the **MARK primary command** is stored as part of the PROFILE options which are maintained individually by file type.

MD - Make Directory

Syntax

MD (File Manager)	<i>directory-name</i>
--------------------------------	-----------------------

Operands

directory-name The desired sub-folder name.

Description

Make Directory. Can be used to create a new sub-folder under the existing File / Path. If the operand contains spaces, it must be enclosed in quotes.

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

MEDIT - Add a File To a Multi Edit Session

Syntax

MEDIT	[<i>filename</i>]
--------------	---------------------

Operands

filename If **filename** is not specified, you will be presented with a Windows open dialog, and asked to specify the file to be added to a MEdit session.

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) |
| ? | Display the current status of the setting. |

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 [DCB - 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.

You can also review [PRESERVE](#) which also affects the length of your data lines.

MODE - Set various Editing Modes

Syntax

MODE	?
	[EDIT BROWSE VIEW]
	[WORD CHAR]
	[CS DS]

Operands

?	Display the current status of the MODE settings.
EDIT	Set the current edit session to EDIT mode, and set EDIT to be the default Mode in the Profile.
BROWSE	Set the current edit session to BROWSE mode, and set BROWSE to be the default Mode in the Profile.
VIEW	Set the current edit session to VIEW mode, and set VIEW to be the default Mode in the Profile.
WORD	Set the default search context to WORD mode.
CHAR	Set the default search context to CHAR mode.
CS	Set the current CHANGE mode to DS .
DS	Set the current CHANGE mode to DS .

Description

The **MODE** command allows you to immediately change certain operating modes of the current edit session.

If **?** is entered the current settings will be displayed.

Setting the basic Edit Mode

The **EDIT**, **BROWSE** and **VIEW** keywords allow you to switch the basic mode of the edit session. i.e. convert an EDIT session to a BROWSE session with **MODE BROWSE**.

Note: You cannot alter the status of an existing Multi-Edit (MEdit) session. You CAN alter an existing Edit session to a MEdit session using the [MEDIT](#) command.

Setting the search context with **MODE WORDS** and **MODE CHARS**

Operands

[WORD | WORDS] |

[**CHAR** | **CHARS**]

Note: **WORD** and **WORDS** are interchangeable, as are **CHAR** and **CHARS**.

Description

These operands alter the normal search context between **CHAR** and **WORD**. 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. 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.

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 **MODE CHARS** or **MODE WORDS** any time you wish on any file tab. This setting is not saved in the profile.

Setting the **CHANGE** data shift mode (**MODE DS** or **MODE CS**)

Operands

[**CS** | **DS**]

Description

Specifies the default Column Shift / Data Shift mode for this edit session. For details see [Effect of **CHANGE/ALIGN** Command on Column-Dependent Data](#) for the significance of this operand. The new default will be saved in the file's Profile.

NDELETE - Delete Lines Where String is Not Found

Syntax

NDELETE	string [PREFIX SUFFIX WORD <u>CHAR</u>] [start-column [end-column]] [line-control-range] [color-selection-criteria] [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** These options describe the “string context” in the same way that a **FIND** or **CHANGE** command does.

PREFIX | **SUFFIX** 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. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

color-selection-criteria A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in ["Color Selection Criteria Specification"](#).

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.

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:

```
DBLANK 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 ]
               [ 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.
<u>NEXT</u>	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.
<u>CHAR</u>	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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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 ]  
           [ MX | DX ]  
           [ TOP ]
```

Operands

search-string	The search string that identifies the lines to 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-	The range of lines which are to be processed by the command. The full

control-range	syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
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 <u>NEXT</u> PREV ALL] [PREFIX SUFFIX WORD <u>CHAR</u>] [line-control-range] [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.
<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-	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are

control-range	discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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** contains 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** contains 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** 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.

NOTIFY - Set Temporary File Notification Level

Syntax

NOTIFY [ALL NONE EDIT RESET ?]
--

Operands

ALL EDIT NONE	Sets the desired temporary notification level
RESET	Sets the notification level back to the permanent level defined in the General tab of the Global Options dialog
?	Display the current status of the setting.

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 [<u>CHAR</u> WORD PREFIX SUFFIX] [start-column [end-column]] [line-control-range] [color-selection-criteria] [MX DX] [ALL] [TOP]
----------------	---

Operands

string	The string operand is required. If 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. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ".
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ".
ALL	All lines in the line range are 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** containing 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 <u>NEXT</u> PREV ALL] [PREFIX SUFFIX WORD <u>CHAR</u>] [line-control-range] [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.
<u>NEXT</u>	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.
<u>CHAR</u>	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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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 [<u>CHAR</u> WORD PREFIX SUFFIX] [start-column [end-column]] [line-control-range] [color-selection-criteria] [MX DX] [ALL] [TOP]
---------------	---

Operands

string	The string operand is required. If provides the string which, if not found, will cause the line to be 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. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ".
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in " Color Selection Criteria Specification ".
ALL	All lines in the line range are 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"](#)

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 in a new SPFLite session.
OPENB SOMETEXT.TXT	Will open SOMETEXT.TXT in a new SPFLite session in Browse mode 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 (New)), 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		CONFIG]

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 Hilights options tab
CONFIG	Open the Global Options dialog and display the Config options tab

C

Abbreviations and Aliases

OPTIONS can also be spelled as **OPTION**, **OPT**

The operands can be specified as minimum unique values. For example **OPT F** is enough for **OPT FILEMGR**, but **OPT S** is not enough as there are 4 operands starting with **S**, you need at least **SU**, **SCR**, **SCH**, and **ST** to distinguish which is wanted.

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** and **KEYBOARD**, you can abbreviate down to one letter, like **G**, **F** and **K**,. but **OPT S** is not enough as there are 4 operands starting with **S**, you need at least **SU**, **SCR**, **SCH**, and **ST** to distinguish which is wanted.

Note: Full descriptions of the options available in each section can be found in:

[General](#)

[File Manager](#)

[Submit](#)

[Screen](#)

[Schemes](#)

[Keyboard](#)

[Status](#)

[Hilights](#)

[Config](#)

PAGE - Set PAGE option

Syntax

PAGE	[ON OFF SCROLL ?]
	[[+ -] <i>offset</i>]

Operands

ON | **OFF** | The desired PAGE mode status

SCROLL

? Display the current status of the setting.

[+ | -] *offset* 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 EOL AUTO 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 }	If specified, the clipboard is copied before, or after, a defined line label.
line-label	<p>If a before/after line label is not specified, the clipboard is copied into the current edit file in one of the following ways:</p> <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 A line command for you on the current line:

Map **Alt-V** to: **(SaveCursor) (LineNo) [A] (Home) [Paste] (Enter) (RestoreCursor)**

Note:

Look closely at the syntax for PASTE, which is:

```
PASTE [ name ] [ { BEFORE | AFTER } line-label ]
[ ERASE ]
```

This means that if you want to use a **line-label** as the point where your data is pasted, you must **also** specify either **BEFORE** or **AFTER**. Neither of those keywords are assumed, and if you use a label but **don't** say **BEFORE** or **AFTER**, the command is illegal. That is true even if it happened to be one of the reserved **.ZF/.ZFIRST** or **.ZL/.ZLAST** labels.

However, you might be thinking, "Why doesn't SPFLite just **assume** I want all the lines inserted at the end if I say **.ZLAST**, or before the beginning if I say **.ZFIRST**?" In other words, we might want these assumptions to be made:

```
PASTE .ZLAST → PASTE AFTER .ZLAST
PASTE .ZFIRST → PASTE BEFORE .ZIRST
```

SPFLite does not make these assumptions, for two reasons:

1. This is how IBM ISPF works, and SPFLite strives to be compliant with ISPF whenever

- possible.
2. The reason **IBM** did this is because when the label is an ordinary one like **.ABC** there is simply no way to determine if the lines go **before** or **after**. It is true that IBM (and SPFLite also) could "cheat" and just make those assumptions anyway, but that might not always be the correct thing to do.

Still, having to say **PASTE AFTER .ZLAST** seems like a lot of typing. Is there anything we could do to make it easier?

Yes. We can define two **SET** assignments, and use them on the command line to shorten this.

First, issue the following SET commands:

```
SET ZL=AFTER .ZL  
SET ZF=BEFORE .ZF
```

Now, when you want to insert **after the end**, do this:

```
PASTE =ZL
```

and to insert **before the beginning**, do this:

```
PASTE =ZF
```

Once these **SET** assignments are made, you can use **PASTE =ZL** and **PASTE =ZF** anywhere a regular command is used, including as a **KEYMAP** definition. Using this **SET** technique is just and concise and easy to as **PASTE .ZL** and **PASTE .ZF** with the added advantage that this way will actually work!

PREPEND - Insert text as start of line

Syntax

PREPEND	string [start-column [end-column]] [line-control-range] [color-selection-criteria] [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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
ALL	All lines in the line range are 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

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.

Created with the Personal Edition of HelpNDoc: [Effortlessly Spot and Fix Problems in Your Documentation with HelpNDoc's Project Analyzer](#)

PRESERVE - Control Handling of Trailing Blanks

Syntax

PRESERVE [<u>ON</u> OFF C ?]

Operands

ON | OFF | The desired PRESERVE status
C
? Display the current status of the setting.

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.

Note: PRESERVE OFF will not remove trailing blanks if they have been marked with any special attributes (like [HILITE](#))

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 use 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.

You can also review [MINLEN](#) which also affects the length of your data lines.

PRINT - Send Selected Lines to the Printer

Syntax

PRINT	[SETUP
	line-control-range
	[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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
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:

SPFLite Print Setup

Current Printer Printer: FinePrint Orientation: Portrait Paper: Letter <input type="button" value="Choose Printer"/>	Current Font Fontname: Raster14 BOLD Pitch: 11.0 Bold: On <input type="button" value="Choose Font"/>
---	---

☐ Use Metric (mm) instead of inches?

<input type="text" value="0.5"/> Page bottom margin (in)	<input type="text" value="0.5"/> Page top margin (in)
<input type="text" value="0.5"/> Page left margin (in)	<input type="text" value="0.5"/> Page right margin (in)

☐ Print band stripes?
 ☒ Print band lines?
 ☐ Print Band Stripe/Line color
 ☒ Print in Color?

☒ Print page headers?

Left-hand header ~f~x	Center header SPFLite	Right-hand header Page: ~#
--------------------------	--------------------------	-------------------------------

☒ Print page footers?

Left-hand footer File Date: ~d ~t	Center footer	Right-hand footer Print Date: ~K(ISODate) ~K(
--------------------------------------	---------------	--

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

<code>~n</code> or <code>^n</code>	The current full filename including path.
<code>~p</code> or <code>^p</code>	The current file path.
<code>~t</code> or <code>^t</code>	The current file time as hh:mm:ss.
<code>~x</code> or <code>^x</code>	The current file extension. For MYDATA.TXT it would be TXT
<code>~k (keyname)</code> or <code>^k (keyname)</code>	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	}	
		{	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 using this Profile.

The Profile data is stored in the normal SPFLite CFG file.

RESET Requests all profile variables be reset to the SPFLite standard values (described below)

COPY Requests all profile variables be copied from another existing file profile.
profile-name

NEW profile-name Some of the profile variables (such as **DCB** and **SOURCE**) affect how SPFLite loads a file into the edit work space. If the profile used for the file type 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 Will bring up the Profile edit dialog for an existing Profile. If several
profile-name 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**

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:

```

EDIT - SPFLite(v3.0.24088.Beta) - e:\GDrive\Test Data\testdata.txt
File Manager testdata.txt
Command > Scroll > CSR

=COLS> 1-----2-----3-----4-----5-----6-----7-----8-----9
=PROF> PROFILE TXT UNLOCKED, ACTION OFF, AUTOBKUP OFF, AUTOCAPS OFF, AUTONAME NONE
=PROF> AUTOSAVE OFF PROMPT, BOM OFF, CAPS OFF, CASE T, CHANGE DS, COLLATE ANSI
=PROF> COLS ON, COMMENTS 1 67 3 99, EMACRO NONE, EOL CRLF, HEX OFF, HILITE FIND AUTO
=PROF> IMACRO NONE, LRECL 0, MACLIB NONE, MARK ON, MINLEN 0, MODE EDIT, NUMTYPE NONE
=PROF> PAGE ON -3, PRESERVE C, RECFM U, SCROLL CSR, SOURCE ANSI, START FIRST
=PROF> STATE ON, SUBARG OFF, SUBCMD OFF, TABS ON, TABBNDS OFF, XFORM NONE, XTABS 0
=WORD> A-Z a-z 0-9 _ $
=MARK>
=MASK>
=TABS> *
=COLS> 1-----2-----3-----4-----5-----6-----7-----8-----9
=BNDS> <+
000001 '----- INCLUDE command
000002 if ClrInclude(i).Len1 = 0 then
000003 aaa,bbb,ccc,ddd,ebee,fff,ggg,hhh,iii,aaa,bbb,ccc,ddd,eee,fff,ggg,hhh,iii,aaa,bbb,ccc,ddd,e
000004 four(five)six

Edit 2024-03-27 14:55 Lines: 185 Cols 1 to 90 Bnds: MAX INS T DS S+ Profile: TEXT APS OF

```

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.

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:

- Enter **PROFILE NEW profile-name**. Here, **profile-name** is the name of the new Profile you want to create.
- A separate pop-up dialog will open where you may select the particular options you require. Click on Done when you are finished.

Your new profile is ready to use. On the first usage, you may then also set the options which cannot be modified by the Profile New dialog, such as TABS, MARK, MASK etc.

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. See ["Working with Profiles"](#) for more details on the Profile Edit Dialog.

Default Profile values set by a PROFILE RESET command

Profile	UNLOCK
ACTION	OFF
AUTOBKUP	OFF
AUTOCAPS	OFF
AUTOSAVE	OFF NOPROMPT
BOM	OFF
CAPS	OFF
CASE	T
CHANGE	DS
COLLATE	ANSI
COLS	OFF
DCB	U CRLF 0
ENUMWITH	1
GLUEWITH	" "
HEX	OFF
HIDE	OFF
HILITE	ON FIND AUTO
IMACRO	NONE
MARK	ON
MINLEN	0
NOTIFY	NONE
PAGE	OFF
PRESERVE	ON
SCROLL	CSR
SOURCE	ANSI
START	OFF
STATE	OFF
SUBARG	" "
SUBCMD	" "
TABS	OFF
XFORM	OFF
XTABS	8
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]
--

Operands

line-control-range

The range of lines which are to be processed by the **PTYPE** command. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

A line range can also be specified by a [C/CC](#) block. Additionally, for the **PTYPE** command, a [M/MM](#) 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.

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.)

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.

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 a Favorite File List

Syntax

RECALL RC [RECENT FILEPATH CONFIG * 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 no operands are present, a default of **RECENT** is assumed. **RECALL RECENT** causes the **Recent Files** File List to be displayed.
- When **FILEPATH** is entered, the most recent file Folder displayed is re-shown.
- When **CONFIG** is entered, the CONFIG display of Instances and Profiles is displayed.
- When ***** is entered, the contents of the ClipBoard will be used as if it were an FLIST file.
- If *list-name* is **FAV**, **FAVORITE** or **FAVOURITE**, it causes the **Favorite Files** File List to be displayed.
- If *list-name* is **FOUND**, it causes the **Found Files** File List to be displayed.
- If *list-name* is **OPEN**, it causes the **Open Files** File List to be displayed. **RECALL OPEN** and [SWAP LIST](#) perform the same function.
- If *list-name* is **PATHS**, it causes the **Recent Paths** File List to be displayed.
- If *list-name* is **FLISTS**, it causes the **FLISTS** display of your File Lists 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.

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**, **FLISTS**, **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

Note: The **RECFM** command has been deprecated, and although it still works as before, you should begin to use the new [DCB](#) command, which replaces RECFM.

RECFM [{ <u>U</u> F V VBI VLI } ?]

Full details of the **RECFM** command operands will now be found in the [DCB](#) command.

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 in the [OPTIONS -> General](#) dialog, which has an upper limit of 25.

RELOAD - Reload Current Edit File

Syntax

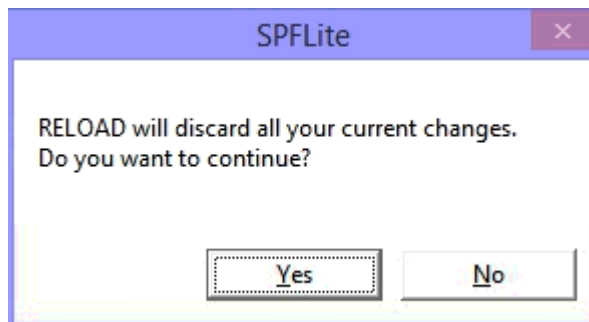
RELOAD

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, View or Browse session.

If you are editing the contents of any of the "special" tabs (CLIP, SetEdit, EFTEdit etc.) you can not issue a **RELOAD**.

When **RELOAD** is performed, the positioning performed by the Profile [START](#) option is not performed.

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.

Note:

In order to maintain compatibility with IBM ISPF, SPFLite uses REN as an abbreviation for RENUM (RENUMBER) rather than RENAME. However, most SPFLite users are likely to use RENAME far more often than the mainframe-oriented RENUMBER command. If you use RENAME frequently, and would prefer to be able to use REN as an abbreviation for RENAME, you can issue the following SET command:

SET ALIAS.REN = RENAME

Or, you can issue a SET command with no arguments to bring up the SET EDIT screen, and enter a line like this:

ALIAS.REN=RENAME

REPLACE - Replace a File

Syntax

REPLACE	[line-control-range] [filename]
----------------	--

Operands

line-control-range	The range of lines which are to be included by the command. The full syntax and allowable operands which make up a line control range is discussed in "Line Control Range Specification" . If no line control range is specified, either via C/CC or M/MM line selection or via the command line, then ALL lines will be selected.
file-name	The name of the file you wish to replace. The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.

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 exist, 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** **DCB** 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 (New), CLIP etc.), a pop-up will appear for you to enter your desired path and file-name.

RESET - Reset

Syntax

RESET (File Manager)	Resets the hidden status of Excluded and/or Forgotten files in a File List
RESET	<pre>[line-control-range] [ALL] [CHANGE] [COMMAND] [WORD] [EXCLUDED] [FIND] [USER] [HANDLE] [HIDE] [LABEL] [RETRIEVE] [SOURCE] [SPECIAL] [STATE] [TAGS] [TRACK] [colorname]</pre>

Operands

line-control-range

The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

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

See the description below.

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.
HANDLE	Clear all current Handles assigned by macros.
LABEL	Clear all current labels markers from the specified range.
RETRIEVE	Reset the primary command Retrieve stack.
SOURCE	Reset the current SOURCE value to ANSI
SPECIAL	Delete all special lines (COLS , WORD , TABS etc.) from the specified range.
SOURCE	Reset the Profile SOURCE value to the default - ANSI
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.
TRACK	Clear the Track stack
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 performs **ALL** the options **EXCEPT** **RETRIEVE**, **LABEL**, **TAG**, **COMMAND**, **WORD**, **HIDE**, **SOURCE**, **TRACK** and **USER** (If global Options "[Default RESET will Revert](#)")

[User Line Status](#)" is not selected,)

If **RESET ALL** is entered:

- it performs a reset of all available options **EXCEPT RETRIEVE, WORD, HIDE and SOURCE**

i.e. the only way to reset these options is to explicitly code them on the **RESET** command.

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.

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 and [CRETRIEV - Conditional Retrieve](#)

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 and [CRETRIEV - Conditional Retrieve](#)

See also [Options - General](#) for setting the minimum Retrieve length.

REVERT - Revert User Line to Ordinary Line

Syntax

REVERT	[string] [CHAR WORD PREFIX SUFFIX] [C] [Q] [T] [start-column [end-column]] [line-control-range] [color-selection-criteria] [MX DX] [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.

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

C	C - Locate the search string within a defined Comment string.
Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).

You may enter more than 1 of **C Q** or **T** to customize the selection. They are tested in an **OR** relationship.

These operands require a valid Profile with Colorization active.

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. The full syntax and allowable operands which make up a line control range are discussed in "[Line Control Range Specification](#)".

color- A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-

selection-criteria	criteria are discussed in "Color Selection Criteria Specification" .
ALL	All lines in the line range are 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 (File Manager)	Repeat the previous FIND command. (No REVERSE option is supported)
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 **RLOC****FIND** to be more useful than **RLOC**. See [RLOC](#)**FIND** - 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

<code>RUN</code>	<code>[.extension]</code>	<code>[cmd-line-operands]</code>
------------------	-----------------------------	------------------------------------

Operands

<code>.extension</code>	Optional operand to override or provide a missing file type extension
<code>cmd-line-operands</code>	Optional operands to be passed on the command line

Description

To properly operate, the file type of the current edit file, must be registered in the system so that Windows can invoke the correct application processor. e.g. a .BAT extension will invoke the system Cmd processor, a .REXX extension would invoke the REXX interpreter.

If the extension of the file in the Edit session is incorrect or missing (as in a CLIP session), then the 1st operand of RUN can be your desired extension which will properly run the file. This operand will be removed and will NOT be passed to the command with any other specified cmd-line-operands.

For a CLIP session, which has no extension, a default for this first operand of **.BAT** will be used.

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.

For this reason, **RUN** will normally only operate on edit sessions where the file is has a single "known name". It will not function in 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 clean up 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	[COND]
-------------	-----------------

Operands

COND **COND** causes **SAVE** to only save edit files which are currently in a modified state. Mainly helpful when used in [Multi-Edit](#) sessions where only a few of the loaded files have been modified.

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 **(New)** 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.

The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.

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] [ALL] [MX DX]
----	--

Operands

line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
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	
			FULL	
			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 EOL AUTO or EOL AUTONL, 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

[Sample SET Edit session](#)

[Using SET to Define SPFLite Options](#)

Syntax

SET	[name	[OFF	 	{ = value }]]
------------	----------	-------------	----------	------------	----------	--------------------	----------	----------

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.

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**.

= value **SET name = value** will assign **value** to SET variable **name**. The value may be a simple word, multiple words, or the value itself needs to be quoted, If so here are the rules for quoting:

1. All values (single or multiple strings) are merged, and **ONE** set of enclosing quotes will be removed.

e.g.

"ABC DEF GHI"	will be treated as	ABC DEF GHI
" 'ABC DEF GHI' "	will be treated as	'ABC DEF GHI'
'ABC "DEF" "GHI"'	will be treated as	ABC "DEF" "GHI"

2. So, if a single sting *value* needs quotes, you must enclose the entire value as follows:

If **'ABC'** is the value, it must be coded as **" 'ABC' "**

3. 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 may be separated by blanks, or not, as you prefer. That is,

```
SET XXX = YYY
```

```
SET XXX=YYY
```

```
SET XXX= YYY      are all equivalent
```

But **SET XXX =YYY** is NOT, since the **=YYY** looks like a request to substitute the current SET value for variable YYY, not what you want.

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.

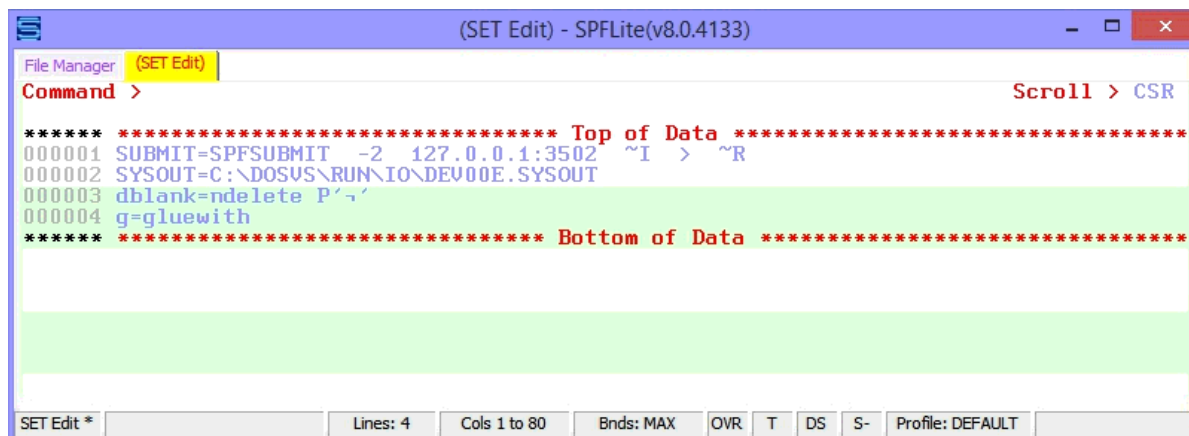
Concatenation

Normally, SET substitution is done in regular 'word' mode, where the SET value replaces the =xxxxx value in the command line. It is possible to concatenate the SET value to following characters. This uses the normal concatenation character |. e.g. If a SET variable was SET ABC = DEF, and the command line contained =ABC|GHI the substituted value would be

DEFGHI.

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)**.



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, 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 the normal **SPFLite.CFG** file, along with most other settings.

Note: As you observe the SET Edit display above, the name and value are normally separated by a single = sign with no spaces between. You may use **xxx=yyy** or **xxx = yyy** as you choose.

Using SET to Define SPFLite Options

SPFLite has several options which are controlled via SET options.

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 can use an alias to modify an existing command. For example, an alias could change an **L** command into an **L TOP** command in order to add **TOP** to any normal **L** command. Or it can simply provide an alias to shorten an otherwise long command or macroname.

You define a command alias by the following **SET** command:

SET ALIAS .name1 [.name2 [.name3]] = command

Note the ALIAS name can contain up to 3 'words' separated by periods. This allows you to

define aliases for phrases. The 'words' refer to the 1st 3 space delimited operands on the command line. So you could have multiple aliases for a single primary command.

Example:

```
SET ALIAS.FIND.A = FIND VALUE1 ALL PREFIX C MX TOP
SET ALIAS.FIND.B = FIND VALUE2 10 20 +RED
```

Which means you can create 'shortcuts' for lengthy commands you use frequently. e.g **FIND A** and **FIND B** become two very different commands.

The **name1[.name2[.name3]]** portion of the SET variable specifies the new alias name you want to define.

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. There are two types of operands:

Simple strings

There are just normal command line operands, The resulting command line will be the **name** you entered, followed by the operands you entered in the SET statement, and then followed by the remaining typed operands on the command line.

Operand Substitution

This feature (Which is allowed **only** for a single name1 variety) allows you to selectively choose or re-arrange operands entered after the **name**. These are symbolic variables of the format **=n**, like **=0**, **=2** etc. Valid numbers are from **=0** to **=9**

The Numbered operands are replaced by the original typed values, where **=0** is the command name, **=1** is the 1st operand, **=2** the 2nd, etc.

Example

```
SET ALIAS.MYCMD = CHANGE =2 =1 ALL
```

This would create a command (MYCMD) which performs a CHANGE command with the operands reversed. i.e. If you entered **MYCMD AAA BBB**, the effective command after substitution would be **CHANGE BBB AAA ALL**.

Note: The operand =variables can be joined (**=1=2** or **=2ABC**), but the full variable names **=ABC** can only be used as separate operands. i.e. you cannot say **ABC=VARDEF** - where **=VAR** is the variable.

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 matched against the 1st 3 operands (depending on the # of names in the ALIAS). If an alias name appears anywhere else, it is an ordinary SPFLite command operand with no special meaning.

Macro Pending

This type of SET statement allows you to flag a particular Macro name as one which should be allowed to generate a "Pending Line Command" message. Normally, Macros would not be allowed to do this. To activate this for a specific macro name enter

SET PENDING.macroname = Y

Macro TRACK activate

Normally Macros do not trigger the creation of TRACK saving, this SET command allows you to activate triggering TRACK support for a particular macroname.

SET TRACK.macroname = Y

AUTOFAV Control

AUTOFAV allows you to specify filenames which match a specific mask format to be automatically added to a specific FLIST. More information on AUTOFAV can be found at "[AUTOFAV to add to FILELISTS](#)" and "[File Lists - FLISTS](#)"

SET AUTOFAV.*.EXT1=EXT1FLIST

MACLIB Specification

MACLIB specifies the name of an alternate MACLIB to be used in place of the normal \Users\You\SPFLite\MACLIB folder. See the [MACLIB](#) command for details.

SET MACLIB.maclibname = full_macrolib_path

MACROMODE Specification

MACROMODE allows you to specify how a macro is to be treated during processing. i.e. Either as a SINGLE line command, or as a BLOCK mode command. Like the difference between a **C** line command, and a **CC** line command.

SET MACROMODE.macroname = BLOCK | SINGLE

PENDING Specification

PENDING allows you to indicate whether a macro is capable of handling other pending line commands or not.

SET PENDING.macroname = Y | N

SORT - Sort the Edit Data

Syntax

SORT	[criteria-1] ... [criteria-5] [line-control-range] [UNIQ MARKUNIQ] [MX DX]
-------------	--

Operands

criteria-1 You may specify up to **five** sort fields. Each criteria is of the following
criteria-2 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. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

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.

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 ]  
          [ C ] [ Q ] [ T ]  
          [ PREFIX | SUFFIX | WORD | CHAR ]  
          [ line-control-range ]  
          [ 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.
C	C - Locate the search string within a defined Comment string.

Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).
	You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.
	These operands require a valid Profile with Colorization active.
line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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:

SHOW C"ELSE"

and press Enter.

This type of selection is called a character string selection (note the **C** that precedes the search string) because it unexcludes the next line that contains the letters ELSE only if the letters are found in uppercase.

SOURCE - Specify Character Encoding

Syntax

SOURCE { encoding-format ? }
--

Operands

encoding-format **ANSI** | **UTF8** | **UTF16LE** | **UTF16BE** | **EBCDIC** | user-format

?

Display the current status of the setting.

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 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

[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 ]
           [ C ] [ Q ] [ T ]
           [ LEFT | RIGHT ]
           [ line-control-range ]
           [ color-selection-criteria ]
           [ 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 <i>split-point</i> .
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 .

<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.
C	C - Locate the search string within a defined Comment string.
Q	Q - Locate the search string within a defined Quoted literal string.
T	T - Locate the search string within plain text (i.e. Not in a Comment or Quoted string).
	You may enter more than 1 of C Q or T to customize the selection. They are tested in an OR relationship.
	These operands require a valid Profile with Colorization active.
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.
<u>CHAR</u>	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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the from-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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**, **RLOC**FIND and **RCHANGE** commands will also apply to **SPLIT**.

Traditionally, **RFIND** (or **RLOCFOUND**) 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 it 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:

- **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	<p>PRIOR requires STATE to be set ON.</p> <p>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.</p>
LABEL	<p>LABEL requires STATE to be set ON.</p> <p>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.</p>
NEW	<p>NEW requires STATE to be set ON.</p> <p>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 <u>automatically</u> placed into the last line of the file when closed.</p> <p>When START is issued with no operands, a message is displayed that shows the current setting of the START option.</p>
?	Display the current status of the setting.

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.

STATE - Control Saving of Edit State Information

Syntax

STATE	[ON OFF MOST FEW DELETE CREATE SAVE ?]
--------------	--

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.
SAVE	This command will create or update the STATE information for the current file. This will be done immediately and will be done regardless of the current setting of STATE (ON, OFF, etc.)
?	Display the current status of the setting.

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.

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
?	Display the current status of the setting.

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 [<i>string</i> OFF ?]
--

Operands

string The *string* 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

? Display the current status of the setting.

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]
	[[argument-list]]
	[DEBUG]

Operands

line-control-range

The range of lines which are to be processed by the command. The full syntax and allowable operands that make up line control ranges are discussed in ["Line Control Range Specification"](#).

[**[argument-list]**]

An optional list of zero or more arguments, enclosed in actual **[]** 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 removed. This cleanup 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:\ONE\TWO.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 CFG** configuration 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 [ON OFF ?]

Operands

ON OFF	The desired TABS status
?	Display the current status of the setting.

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.

If no operand is entered, the command wil 'toggle' the setting between ON and OFF. The resulting setting will be displayed in the confirmation message.

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 | SET ]
          [ FIRST | LAST | NEXT | PREV | ALL ]
          [ PREFIX | SUFFIX | WORD | CHAR ]
          [ C ] [ Q ] [ T ]
          [ line-control-range ]
          [ color-selection-criteria ]
          [ 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 TOGGLE SET	Specifies the action	
	ON	Will set the specified tag on the line.
	OFF	Will clear the specified tag on the line.
	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

If **:tagname** is NOT specified, it will clear **any** existing tagname on the line.

If **:tagname** is specified, it will clear the tagname **only** if the existing tag on the line matches the one specified.

If **:tagname** is specified, and no existing tagname is present, it will assign **:tagname** to the line.

SET

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.

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.

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.

C - Locate the search string **within** a defined Comment string.
Q - Locate the search string **within** a defined Quoted literal string.
T - Locate the search string **within** plain text (i.e. Not in a Comment or Quoted string).
 You may enter more than 1 of **C Q** or **T** to customize the selection. They are tested in an **OR** relationship.

These operands require a valid Profile with Colorization active.

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
<u>CHAR</u>	.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. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
color-selection-criteria	A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in "Color Selection Criteria Specification" .
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]
	[ALL]
	[MX DX]

Operands

line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in " Line Control Range Specification ".
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.

Created with the Personal Edition of HelpNDoc: [Revolutionize Your Documentation Output with HelpNDoc's Stunning User Interface](#)

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]
	[ALL]
	[MX DX]

Operands

line-control-range	The range of lines which are to be processed by the command. The full syntax and allowable operands which make up a line control range are discussed in "Line Control Range Specification" .
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 ]
           [ C ] [ Q ] [ T ]
           [ start-column [ end-column ] ]
           [ line-control-range ]
           [ color-selection-criteria ]
           [ MX | DX ]
           [ 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.

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

C - Locate the search string **within** a defined Comment string.

Q - Locate the search string **within** a defined Quoted literal string.

T - Locate the search string **within** plain text (i.e. Not in a Comment or Quoted string).

You may enter more than 1 of **C Q** or **T** to customize the selection. They are tested in an **OR** relationship.

These operands require a valid Profile with Colorization active.

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. The full syntax and allowable operands which make up a line control range are discussed in ["Line Control Range Specification"](#).

color-selection-criteria A request for selection based on the highlight color of the search-string. The full syntax and allowable operands which make up a color-selection-criteria are discussed in ["Color Selection Criteria Specification"](#).

criteria

ALL	All lines in the line range are 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 [OPTIONS -> General dialog](#). a maximum of 25 **UNDO** levels can be set.

Note: The effect of an **UNDO** can be reversed by the [REDO](#) command.

VIEW - View a File in Browse Mode

Syntax

VIEW	[file-name]
	[.Profile-name]
	[%imacro-name %ON %OFF %NONE]
	[/XForm-macro /ON /OFF /NONE]

Operands

file-name	<p>The name of the file to be viewed.</p> <p>The filename may contain system variables (%xxx%) if desired, they will be substituted from the System variable settings.</p> <p>If the file-name does not exist, VIEW will be cancelled.</p>
.Profile-name	<p>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)</p>
%imacro-name	<p>This optional operand allows you to specify an IMACRO (a macro to be run immediately after the file is loaded). If an IMACRO is specified in the file's Profile, this imacro-name will override it.</p> <p>You can use /OFF or /NONE to nullify an IMACRO specified in the Profile</p> <p>You can use /ON to activate an IMACRO in the profile which is save in OFF mode.</p> <p>See Profile IMACRO for details.</p>
/XForm-macro	<p>This optional operand allows you to specify an XFORM (a macro to allow access to non-standard file types). If an XFORM macro is specified in the file's Profile, this XForm-macro will override it.</p> <p>You can use /OFF or /NONE to nullify an XFORM macro specified in the Profile</p> <p>You can use /ON to activate an XFORM macro in the profile which is saved in OFF mode.</p> <p>See Profile XFORM for details.</p>

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 (New)) 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.

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 menu".

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](#).

XFORM - Specify a File Transform Macro

Syntax

XFORM	[<u>NONE</u>] Macro-name] [ON OFF ?]
--------------	--

Operands

NONE Will nullify any existing XFORM macro-name.

Macro-name The Macro-name to be used for external file reading / writing.

If **ON** or **OFF** are specified, they indicate:

ON - The macro is activated and will be used on each file load/save.

OFF - The macro name is saved, but is "dormant", it will not be used on file load/save.

Note: the ON/OFF state can be over-ridden at file open time by adding a **/ON** or **/OFF** to the [EDIT](#), [BROWSE](#) or [VIEW](#) command. In File Manager, the **/ON** or **/OFF** can be entered as a line command.

? Display the current status of the setting.

Description

The **XFORM** command allows you to specify a Macro which will be effectively "take over" the reading and writing of data to/from the external file. This allows you to create a macro which could provide access to external files which are not normally accessible to SPFLite.

This is an advanced concept, full details are provided in "[Working with XFORM supported files](#)".

NONE If **XFORM** is issued with no operands, it will simply display the current **XFORM** setting. Therefore, in order to nullify an existing **XFORM** command string, **NONE** is used to represent the "" condition.

Actions when only some operands are entered

- When only Macro-name is entered
 - If no current Macro-name, **ON** is assumed
 - If there **is** a current Macro-name, the existing **ON/OFF** state is assumed.
- When only **ON/OFF** is entered
 - If no current Macro-name, an error message is displayed
 - If there **is** a current Macro-name, it is kept and the new **ON/OFF** setting saved.
- When **NONE** is entered, any other operands are ignored, and any existing Macro-name is nullified.

XTABS - Control Incoming Tab Characters

Syntax

XTABS	[n ?]
--------------	-------------------------

Operands

n	The desired size of the tab spacing
?	Display the current status of the setting.

Description

The **XTABS** (or alias **XTAB**) 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.

Using **XTABS** as default for Data Shifting commands.

The data shifting commands - ((())) > > > < < < [[[and]]] use the default value specified in Options - General. It is possible to request the Data Shift commands to use the XTABS value (if specified) for this default. Refer to [Options - General](#) for details.

SPFLite Macro Support

Macros Introduction

The Edit Macro support in SPFLite provides a means to automate complex or repetitive editing tasks, and to create functions not available with SPFLite's existing built-in edit commands.

Macro support is available in both normal Edit / Browse / View sessions, and in the File Manager tab as well. This provides the ability to extend the possible activities in the both types of tabs by using the powerful functions provided by the thinBasic script engine. thinBasic provides a wide range of file related functions, it is well worth your time to do some exploratory reading.

Operating Modes

Whether a macro is an FM related macro, an Edit macro, a Line command macro or a Primary command macro, the structure and support provided to macro writers is as similar as we could make it.

But there **are** some differences in support for FM related macros and Edit/Browse/View related macros.:

- Some functions are **only** supported in the FM environment, and they all start with FMxxxxx to easily identify them. These FM related macros will not function in a normal Edit tab.
- Similarly, many of the old macro functions will not operate in the FM environment. If a macro attempts to use a function in the 'wrong place', the macro will be terminated with an error message identifying which function is in error. The [Function Overview](#) section indicates which functions will operate in which environment.
- Many of the existing functions, like those which access the environment variables, the global storage areas etc. will of course work in either environment.
- The timing of macro invocation differs as well - **This is an important distinction!**:
 - In an Edit tab, line macros are processed **before** primary command macros or primary commands.
 - In an FM Tab, **Primary** Macros are executed first, followed by **Line** command macros, followed by **built-in** line commands.

Primary commands can utilize operands coded on the command line following the macro name and can use any of the supported methods for assisting in the parsing of the operands

In the Edit tabs, primary commands can also access and reference built-in line commands. e.g. in Edit, a primary macro can reference line ranges marked with **CC/CC** markers.

In the FM tab, primary macros can examine and alter or delete line commands that may already be present. They can also add line commands for execution if desired.

In Edit mode, Line command macros can utilize either single line or block mode line marking.

Being line commands, these macros usually have short names. Whether a line command is considered a single line or a block mode line command is discussed in the next section [Macro Format and Structure](#)".

For Line command macros in Edit, because of their nature, there are some additional restrictions on usage

- Line command macros cannot be entered at the same time as commands on the command line.
- Only one line command macro at a time may be entered; and only one instance of that macro. e.g. for the macro **AX** you could not enter a series of individual **AX** line commands. A pair of block mode **AXX** lines is considered **one** instance.
- Line command macros can not be entered at the same time as other built-in line commands which are considered 'source' line markers, like **CC** or **MM**, since line command macros are themselves always treated as a source range.

Script Engine

The script engine chosen to support the macro facility is called **thinBasic** (www.thinBasic.com).

Don't let the word "**thin**" fool you. **thinBasic** is no "toy" language. This is a very capable script engine written in, and modeled after, PowerBasic (www.PowerBasic.com). Since SPFLite itself is written in PowerBasic, the interface between SPFLite and **thinBasic** is quite straightforward, which makes possible a highly efficient implementation. All of our test scripts have run very quickly, and you should see similar good performance as you write your own scripts.

Being a BASIC variant, users with experience in typical programming or scripting languages should recognize most of the **thinBasic** syntax. Many sample macros have been provided to help you get started.

As you will quickly notice, keywords and syntax in **thinBasic**, like SPFLite, are case-insensitive. When you read the documentation and sample code that follows, the choice of capitalization you see is simply a style choice. You are free to capitalize (or not) any way you see fit.

thinBasic components are included in the normal SPFLite install, so no separate install of **thinBasic** itself is required. If however you would like to explore **thinBasic** as a normal scripting language, then you should obtain their full install package and install it. We recommend, as of this writing, you install release 1.10.7.

If you go that route, be prepared for a lot of reading. The full **thinBasic** help document is included in the basic SPFLite install, and can be accessed from the Start Menu SPFLite program group, or from within SPFLite itself by entering `HELP THINBASIC`. You will be surprised how much is in it.

This document is organized into the following sections.

The [SPFLite interface](#). This contains the reference documentation for each of the SPFLite interface calls. It also contains tutorials on interface techniques: How to handle command parameters, how to search for and manipulate text lines, how to create a new Line command, how to control the edit cursor location, etc.

[Sample macros](#). Several of the included sample macros are shown with full comments.

[thinBasic essentials](#). This is a simplified introduction to **thinBasic**. It is not a complete

and definitive description of all **thinBasic** capabilities and functions, but should be enough to get you by. For more than that, refer to the **thinBasic** Help file as described above.

Working with the thinBasic Interpreter

As interpreters go, **thinBasic** is quite sophisticated. However, it remains a true interpreter. What that means to you is that it does not perform a "full program parse" before beginning to execute your script. Instead, it reads, parses and executes a line at a time. So, if you have a macro that's 10 lines long, and there is an error in line 10, it will perform the first 9 statements, then halt with an error-detected message. You will have to correct your errors one at a time until you get past this.

In practical terms, you may find it beneficial to write larger macros in pieces, starting small and testing a bit at a time, and adding more code as you go along, to minimize any problems from syntax errors you might introduce into your code.

You can also develop "pieces" of **thinBasic** code and **#INCLUDE** them into your macro. Doing this is a way to 'modularize' your code, so that you can deal with tested pieces of **thinBasic** code that you know will work correctly. This is a technique that might help you if your macro is quite large. See the **thinBasic** documentation regarding the [#INCLUDE statement](#). **Note:** At this point, the **#INCLUDE** statement does not appear to handle unqualified file names consistently. Until this is resolved, it would be prudent to fully qualify the included filenames.

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Macro Format and Structure

Contents of Article

[Macro Names vs Actual Command Names](#)

[Macro Prototype](#)

[Macro Code structure](#)

Introduction

The format of a macro is a standard text file of thinBasic source statements stored in the SPFLite MACROS folder.

For most users, the full path of this folder will be:

C:\Users\username\My Documents\SPFLite\MACROS

The macro should be named **macname**.MACRO, where macname is the name with which you will be invoking the macro.

The string "**macname**" is also what is returned by the [Get_MacName\\$](#) function. As with everything else in SPFLite, macro names are case insensitive.

Note: Because the macro's name gets parsed as part of a primary command or line command, you cannot use all legal Windows file name characters in a macro name, but should generally limit them to letters, digits and underscore.

Further, for line-command macros, the name must be even more restricted; the name cannot

contain any digits, because they will be confused with numeric line-command operands. For line commands, the macro name should normally only contain letters to avoid problems. A few special characters were found to be acceptable, such as \$ and @. We have not exhaustively tested every possible character, but if you can create a legal Windows file name with it, it will usually be valid as a macro name, except for the characters **+ - * ? : . /** and ****.

Macro Names vs Actual Command Names

While macro names are usually chosen to be unique, you may name a macro the same as a normal, built-in SPFLite command. e.g. You could name a macro **ALIGN**. If you do so, your macro will replace the normal command. That is, there is no more access to the normal **ALIGN** internal command. **EXCEPT** - the normal internal command can always be accessed from within a macro via the [**SPF_CMD**\(*command-name*\)](#)

Edit mode Line Macro names and Block Mode

For single line macros (e.g. do THIS to a single line) there is no problem, the line command = the macro name.

When you want to create a macro that can act as a block mode command (like **CC/CC**) the following methods are used:

- Repeating the last character of the line command. Using the same example as above, the block form of **AX** would be requested by entering **AXX/AXX**, and the macro still stored as **AX.MACRO**. This applies also to longer macro names. For example the block mode version of a macro called **BOX** would be **BOXX**, or for a macro called **PRINT** it would be **PRINTT**.
- Specifically tell SPFLite in what mode a macro name is to be treated. This is done by issuing a SET command of the format:

SET MACROMODE .macname = BLOCK | LINE

If a **SET MACROMODE** has been issued for a macro name, the convention about repeating the last letter does not apply. Any length or format of macro name can be set unconditionally to a specific processing mode.

The only requirement SPFLite imposes on the format of a macro file is the macro prototype (or header), which must be the first line in the macro. Like everything else in SPFLite, the prototype is generally case-insensitive, unless it has default operands used as string values in FIND or CHANGE commands, for instance.

Macro Prototype

The first line of a macro must always be the macro prototype. This is a simple thinBasic comment statement of the format:

' macname.MACRO [def-operand-1 def-operand-2 ...]

The **.MACRO** part of the prototype is required. **macname** itself is optional, but should normally be the name of the macro. The brackets shown mean the list of operands is optional; don't actually code brackets here.

Currently, SPFLite does not demand that **macname**, if coded, match the actual file name of

the macro. However, for documentation purposes, it is best that you **do** make the name agree, just to keep things straight as you develop and use your macros. It is possible that a later release of SPFLite will require that the prototype name agree with the file name, so it's best to make them agree now.

You may optionally enter default values for macro operands (if your macro uses command-line operands). These are simple space-delimited strings which provide defaults if the relative operand number is not overridden when the macro is called. For example, given the following prototype:

```
' sample.MACRO  aaa  bbb  ccc
```

If the macro were invoked with the primary command line as **Sample** with no supplied arguments, then if the executing macro requested the number of operands via **Get_Arg_Count** it would receive 3. **Get_Arg\$(1)** would be **aaa**, **Get_Arg\$(2)** would be **bbb**, and **Get_Arg\$(3)** would be **ccc**.

More details on retrieving macro operands and working with them will be found in [Accessing Command Line Operands](#).

Macro Code structure

As described above, an SPFLite macro program is essentially "open code" in **thinBasic** and has no "structure" per se. By that, it means there is no MAIN Sub/Function, execution just starts at the first line and continues on.

As a result, there is, by default, no built in mechanism for SPFLite to 'pass in' the operands. This is easily correctable by making the mainline code into a thinBasic SUB routine. If this is done, then the command line operands can be made available as passed parameters.

Before making this decision you should evaluate how you wish to handle your declared variables. In Open Code, all DIM'd variables are effectively Global and are 'visible' throughout your program. If you convert to structured mode and make the mainline a SUB, then DIM'd variables are only 'visible' throughout the mainline SUB. If you use additional SUB routines, then they will need to be passed any working data they require.

More details of this support will be found in [Accessing Command Line Operands](#).

Locating Macros and Include files

Contents of Article

[Automatic modification of the #INCLUDE statement at run time](#)

[Automatic insertion of #INCLUDEDIR directive at run time](#)

[Support for SET names in #INCLUDE statements](#)

[Using relative paths for included files](#)

Introduction

As stated previously, macros are stored in the SPFLite MACROS folder, which will be

C:\Users\username\My Documents\SPFLite\MACROS

If you are writing a simple, one-file macro, that is all you need to know. If you are writing a large macro in a modular fashion, or are writing a series of related macros having code in common, you would benefit from using **#INCLUDE** statements to organize your macro code into smaller and more manageable pieces.

The macro engine we use, called **thinBasic**, allows for **#INCLUDE** statements. However, in earlier versions of SPFLite, these were not convenient to use. According to the **thinBasic** documentation, an **#INCLUDE** statement with no explicit path is searched for in the following order:

1. the current script path
2. \thinBasic\inc
3. any #INCLUDEDIR paths in effect

When **thinBasic** is integrated into other software like SPFLite, step 2 becomes "\Program Files\SPFLite\inc". That can be a problem. Putting changeable user data under the Program Files directory is not a good idea, and starting with Windows 7, you may need Administrator privileges to do that. Since SPFLite has already set aside the MACROS folder, it makes more sense to be looking for included macro files there.

Automatic modification of the #INCLUDE statement at run time

To make this process simpler, SPFLite now will modify your **#INCLUDE** statements, so that if it detects a file name that has **no** "path qualifier" at the beginning of the name, the location of the SPFLite MACROS folder is added to the file name. That way, any "simple" names you use on **#INCLUDE** statements will automatically be searched for in the same place that your main macro is located.

This "modification" to your **#INCLUDE** statements only involves altering an internal copy of these statements in memory, as they are being processed. Your **#INCLUDE** file itself on disk is not changed. SPFLite will never modify your macro or include files.

In deciding when to apply this modification, the beginning of the file name is examined. If that name begins with / or \ or . or if the second character of the name has a : colon, it is assumed to have an "explicit path", and the file name is not modified.

Note that **included** file names do not have to end with **.macro** but can be anything you wish.

A common extension is **.inc** but that is entirely up to you.

This automatic modification of **#INCLUDE** statements is done only in the main macro file, **not** in nested files included by the main macro file. That is because these are read by thinBasic directly, and SPFLite does not "see" them.

For instance, if you write an **#INCLUDE** statement like this,

```
#INCLUDE "myfile.inc"
```

The name **"myfile.inc"** does not begin with **/** or **** or **.** or a drive letter like **c:.** SPFLite will see that there is no path qualifier on the file name, and will internally change the statement to something like this:

```
#INCLUDE "C:\Users\username\My
Documents\SPFLite\MACROS\myfile.inc"
```

Automatic insertion of **#INCLUDEDIR** directive at run time

Like many languages, **thinBasic** allows **#INCLUDE** statements to be nested. When you do this, **thinBasic** itself reads those nested files. SPFLite is not involved in that process, and does not read or see nested **#INCLUDE** statements. Because of this, the special handling described above is **not** done for such nested includes.

To help with this situation, SPFLite will automatically insert an **#INCLUDEDIR** directive just prior to the first **#INCLUDE** statement it detects in your main macro file (again, this is done internally - not to your macro file on disk). The "created" **#INCLUDEDIR** directive will specify the path to the SPFLite MACROS folder, so that nested includes will be searched for there.

The **#INCLUDE** and **#INCLUDEDIR** statements are documented in the **thinBasic** Help. Issue a **HELP BASIC** command from SPFLite to bring up this Help.

Suppose you wanted you write your own **#INCLUDEDIR** directives as well. Will that work? Yes. You need to specify any desired **#INCLUDEDIR** directives prior to your first **#INCLUDE** statement. This will cause **thinBasic** to look for included files in the locations you specify, in the order you write them. SPFLite will simply ensure that, if an included file is not found in any of **those** locations, it will look in the SPFLite MACROS folder as a "last resort" search location.

This process should be taken into account if you are planning on creating some complicated search-path for a very involved macro or set of macros, that might also involve multiple versions of files (like, "test" vs. "production" versions). If you put a copy of an included file in the SPFLite MACROS folder, **and** you issue an **#INCLUDE** statement that, for whatever reason, does not find this file elsewhere, it **will** look the file in the SPFLite MACROS folder as the default search location. If that is not what you want, you should not put files in that default folder, but only in the explicit locations you need them to be in.

If your macro never issues an **#INCLUDE** statement, it will not insert this **#INCLUDEDIR** directive.

Support for SET names in **#INCLUDE** statements

You may wish to organize your included files into folders other than the default SPFLite MACROS folder, but you also would probably not like to use absolute path names embedded in your **#INCLUDE** statements. SPFLite now allows you do to this.

If you write an **#INCLUDE** statement like this,

```
#INCLUDE "=abc\myfile.inc"
```

SPFLite will look up the SET name you specify ("abc" in the example) and substitute the value of the SET variable into your **#INCLUDE** statement. SET names are defined by the SET primary command.

Suppose you issued the following SET primary command:

```
SET abc = "C:\mypath"
```

Then, when you run your macro that has the **#INCLUDE** statement above, the SET name is detected and looked up, and its value is substituted prior to thinBasic processing it. So, the file that actually gets included is:

```
#INCLUDE "C:\mypath\myfile.inc"
```

If your SET name "abc" happens to be undefined when you run your macro that has the **#INCLUDE** statement above, SPFLite will not attempt to modify it. You will then get a syntax error message from thinBasic that the name "**=abc\myfile.inc**" could not be found.

This modification of **#INCLUDE** statements to substitute SET name values is done only in the main macro file, **not** in nested files included by the main macro file. That is because these are read by **thinBasic** directly, and SPFLite does not "see" them.

Using relative paths for included files

If you want to organize a group of included files into its own folder, but you'd like to keep the process simple, you can use **relative paths** to accomplish this.

Suppose you wanted a set of included files to be stored under a folder called "**xyz**". To do this, create a folder with the name "**xyz**" under the SPFLite\Macros directory. Then, you would write your **#INCLUDE** statements like this:

```
#INCLUDE "xyz\myfile.inc"
```

The name "**xyz\myfile.inc**" does not itself begin with / or \ or . or a drive letter, even though there is a backslash after the "**xyz**" part. SPFLite will see that there is no path qualifier on the file name, and will internally change the statement to something like this:

```
#INCLUDE "C:\Users\username\My  
Documents\SPFLite\MACROS\xyz\myfile.inc"
```

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

Accessing Macro Command Line Operands

Contents of Article

[Basic operand access](#)

[SUB operand access](#)

[A note about macro parameter data type](#)

[Full-parse access](#)

[How to access the parsed out operands](#)

[How to specify the macro command syntax](#)

[How to specify a variable number of values](#)
[How to specify optional values](#)
[How to validate tag and line references](#)
[How to specify keywords](#)
[Example: Putting SPF Parse all together](#)
[A Simple Macro Demo in All Three Command Operand modes](#)

Introduction

Except for the simplest of macros, command line operands are necessary to tailor the actions performed by your macro. So, retrieving and 'sorting out' what operands have been entered is a necessary requirement for most macros. The SPFLite macro support provides three methods for handling the operands. Which is best for you? The deciding factors are the number, order and complexity of the operands, whether any are optional, and whether any are keywords which possibly are part of a "keyword group". The chart below will help you by summarizing the differences in the three levels of support.

FM Line Command Macro differences

When a macro is issued on a File Manager line command, SPFLite will **insert** a numeric operand immediately after the macro name to indicate which line of the FM list the macro was entered on. e.g. If **MYMACRO AA BBB CCC** was entered on line 15 of the display, the macro would be invoked as **MYMACRO 15 AA BBB CCC**. That is, the line number is always the first operand, all others are shifted right.

It is important to understand how the command line operands are handled. SPFLite does an initial parse of the command line operands, which are delimited by spaces or quotes as usual, and stores them into a table, which can be accessed with the [Get Arg\\$](#) and [Get Arg Count](#) functions.

These operands are not initially categorized or sorted in any way. You would simply "get" them, using the functions described next.

There are three types of macro operand access, as follows:

Basic Access

- The number of operands is available via [Get Arg Count](#)
- Each operand can be accessed as [Get Arg\\$\(n\)](#), where **n** is the operand position on the command line; the first operand after the macro name itself is considered operand number 1. Operands are returned as STRING values, as indicated by the \$ in the function name.
- If local variables are needed for processing the operand values, they must be declared in a DIM statement and assigned the [Get Arg\\$\(\)](#) value. These two steps can be combined if you wish, so that you could write **DIM FIRST_ARG AS STRING = GET_ARG\$(1)** or these can be done as separate steps.
- All validation and handling of the operands is the responsibility of the macro code you write.

SUB Operand Access

- In creating the SUB structure, the operands are assigned, from left to right, to the arguments defined in the SUB statement. **NOTE:** You must specify at least one operand. If you have no operands, simply insert DUMMY AS STRING for the operand.
- Operands are accessed using the variable names used in the SUB statement, making the code much more readable. e.g. if the first operand of the SUB statement were coded as **start-line AS STRING** it makes code much more readable to code **IF start-line = xxx** than to code **IF Get_Arg\$(1) = xxx** as in the Basic Access mode.
- Local variables do not need to be declared with a DIM statement for storage of the data.
- All validation and handling of the operands is the responsibility of the macro code you write.

Full Parse Access

- Most built-in SPFLite primary commands utilize 'order-independent' operands. For instance, you can say **CHG FRED BILL ALL** or **CHG ALL FRED BILL** or **CHG FRED ALL BILL** and SPFLite will figure out what you mean. Additionally, many keyword operands have multiple aliases, such as PREFIX, PRE, and PFX. Handling all these variations of operand formats in a macro, using your own code and positional operand access methods, would be difficult and error-prone.
- The function **SPF_Parse** does this 'sorting out' for you, identifying all the various operand types, keywords, and keyword aliases. This allows your macro to provide flexible operand handling with much less effort.
- The main requirement is for you to specify with the **SPF_Parse** function the detailed syntax requirements of your command. This involves informing SPFLite as to how many labels, tags, numbers, string values and keywords you expect to be used.
- **SPF_Parse** is normally be invoked at the beginning of the macro, and, if no scanning errors are reported, all the command line operands will have been categorized and set up for easy retrieval when you need them. Since this is an ordinary function like others provided by SPFLite, you are free to call this function wherever it is most appropriate for your logic.
- The categorization supports operand types such as:
 - Line references (like .AA .BB .123)
 - Tag references (like :DD :S1)
 - Text literals (like ABC C 'DEF' K5)
 - Numeric literals (like 10 20 30)
 - Keywords (like ALL PREFIX | SUFFIX |

WORD)

This support includes handling of aliases and mutually-exclusive keywords.

- Optional validation of Line References and Tag operands can be performed for you. Invalid (undefined) label or tag operands will cause SPF_Parse to report a syntax error. Because this process is optional, you can decide if you expect your label or tag operands to already exist, or if they might be new labels or tags that the macro itself would define in the course of its operation. To validate a label using macro code, you can try converting a label to a line pointer using [Get_Lptr](#), where a non-zero result means the label is valid. To validate a tag, you can try issuing a command like [SPF_CMD](#)("LOC :ABC FIRST") and if you get RC=0 the tag should be valid.
- Parameter validation that is specific to your macro's application must be done by your code.

Basic operand access

Using basic access, macro operands are accessed positionally by their position in the command string, from left to right. The total number of operands available is available by calling the [Get_Arg_Count](#) function. Each individual operand is obtained by calling [Get_Arg\\$\(n\)](#) where **n** is the operand number; the first operand after the macro name itself is operand number 1.

The macro may have default operands specified by the macro prototype line (see [Macro Prototype](#)) For example, given the following prototype:

```
' sample.MACRO  aaa  bbb  ccc
```

If the macro were invoked with the primary command line as **Sample** with no supplied arguments, then the [Get_Arg_Count](#) function would return 3. [Get_Arg\\$\(1\)](#) would be **aaa**, [Get_Arg\\$\(2\)](#) would be **bbb**, and [Get_Arg\\$\(3\)](#) would be **ccc**.

If the macro were invoked with the command **Sample RED GREEN BLUE YELLOW**, then the [Get_Arg_Count](#) function would return 4. [Get_Arg\\$\(1\)](#) would be **RED**, [Get_Arg\\$\(2\)](#) would be **GREEN**, [Get_Arg\\$\(3\)](#) would be **BLUE**, and [Get_Arg\\$\(4\)](#) would be **YELLOW**.

If the macro were invoked with the command **Sample RED**, then the [Get_Arg_Count](#) function would return 3. [Get_Arg\\$\(1\)](#) would be **RED**, [Get_Arg\\$\(2\)](#) would be **bbb**, and [Get_Arg\\$\(3\)](#) would be **ccc**.

All other validation of the operands is the responsibility of the macro itself.

SUB operand access

If you choose to code your macro in a more "structured" design, then you can have the command line operands provided to you as passed parameters to your mainline SUB

subroutine. To do this, certain considerations which must be met.

To convert your macro to a structured format, you must place an initial SUB subroutine header immediately after the macro prototype line, as described in [Macro Prototype](#). If you do this, and you normally use your macro from the primary command line with some fixed maximum number of parameters, SPFLite will assign these for you just like any other SUB subroutine would receive them, so that calls to **Get_Arg\$** will not be needed.

This can be a quite useful feature, but it requires you to follow some fairly strict rules to take advantage of this:

- The initial **SUB** subroutine follows all the rules of **thinBasic** syntax, and must be ended by an **END SUB** statement.
- The SUB keyword must appear immediately following the SPFLite macro prototype line, meaning that the initial SUB keyword must be on line 2 of the MACRO file and nowhere else, and no comments may appear anywhere on the SUB statement.
- The initial SUB line may be continued on multiple lines if needed, by placing an _ underscore at the end of each continued line except the last one, as required by standard **thinBasic** line-continuation rules.
- The type of each parameter to the initial SUB must be individually declared with **AS STRING**. No other data type will work. **Note this point carefully. If you fail to do this, your macro will not work correctly.**
- In the body of the initial SUB subroutine, you can optionally issue the HALT statement anywhere you wish to end the macro. Otherwise, an implied HALT will be automatically issued when the subroutine issues an EXIT SUB, or passes through the normal END SUB.
- The initial SUB subroutine may have any valid name you wish.
- The initial SUB subroutine name need not match the name (if any) on the macro prototype header that appears on line 1 of the MACRO file, or the name of the MACRO file itself. However, for clarity and documentation purposes, it is recommended that you **do** make these names match. Future versions of SPFLite may or may not enforce such naming agreement.
- The parameters defined on the SUB statement are provided by merging the actual command line operands with the default operands which may be provided on the [Macro Prototype](#) header. When no value exists in either location, a null string "" is passed as the parameter.
- SUB subroutine statements can only define a fixed number of parameters. If you wish to access parameters that were specified on the primary command line beyond these, or need to handle a variable number of parameters, the **Get_Arg\$** function and other related functions are available and can still be used as documented, even though you used an initial SUB subroutine.
- When you write a macro with an initial SUB subroutine, and the macro is launched as a line-command macro, only the defaults on the macro prototype header will be passed, and if the prototype has no defaults, all of the parameters passed to the SUB will be empty (null) strings.

NOTE: Unless you strictly follow the SUB formatting rules, SPFLite may not detect that you

have put an initial SUB subroutine in your macro, and it will not get called.

We have to perform a clever bit of "magic" to pull this off. SPFLite has to do a "read-ahead" and "peek" at your macro, and quickly analyze it to determine if you actually **have** an initial SUB subroutine like this. When it finds one, it inserts a call to that SUB, passing the appropriate parameters as needed. To do that, and do it reliably, you **must** ensure that the format of the SUB line strictly adheres to these rules.

In the initial SUB subroutine, you are free to call other subroutines and functions as desired. These can be built-in functions, functions imported by USES statements, or additional SUB and FUNCTION routines that you write yourself.

As with Basic Operand Access, validation of the command line operands is still entirely up to the macro code to perform.

A note about macro parameter data type

It is important that you follow rule noted above to define all operands AS STRING. For example, this macro,

```
' sample.macro abc def
SUB sample (arg1,arg2,arg3 AS STRING)
    ' ... statements
END SUB
```

is **incorrect**. The reason is that, as written, the **AS STRING** clause does **not** apply to all three parameters, but **only to the last one**. This is simply a **thinBasic** syntax rule, and we have no control over it. However, you needn't worry about forgetting this rule, SPFLite will verify this has been done properly and reject the macro if it breaks the rule.

What actually happens here is that when you have parameters written like "**arg1,arg2,**" with no type information specified, **thinBasic** treats these as parameters of type **VARIANT**. It's too complicated to discuss here, except it's just not what you want. SPFLite will only pass you parameters as strings anyway, so you have to cooperate and do it the right way.

The correct way to write it is like this:

```
' sample.macro abc def
SUB sample (arg1 AS STRING, arg2 AS STRING, arg3 AS STRING)
    ' ... statements
END SUB
```

If you like, you can put each parameter on a separate line, for readability and maintenance purposes, using line continuation like this:

```
' sample.macro abc def
SUB sample (arg1 AS STRING, _
            arg2 AS STRING, _
            arg3 AS STRING)
    ' ... statements
END SUB
```

Full-parse access

This support co-exists with all the Basic Access and SUB Operand Access methods for accessing macro operands. It make **no changes at all** to the **Get_Arg\$** and **Get_Arg_Count** functions, or to the support for SUB operand handling.

You are free to choose whatever method suits you best, and you also can combine these methods. For instance, your macro operands might have a "simple" format that does not need to be parsed, and also a more complex format. You can choose to do the full parse if you determine that the simple format is not present.

The idea is that a call to **SPF_Parse** is made, passing a set of parameters that describe the syntax of your macro's operands. Using these parameters, and the initially-created contents of the **Get_Arg\$** data array, **SPF_Parse** will 'sort out' and validate the macro operands according to your specifications. If all is well, it returns an RC=0, and makes all those parameters available for use. If a parsing or validation error is found, it will issue an RC=8 along with an error message which you can obtain with **Get_Msg\$** and display to the user.

Validation consists of the following:

- Depending on the optional validation flags, verification that the number of operands entered of that type meets the specification. Using flags, you can control whether an operand count represents an exact count, or is an upper limit (where the lower limit is zero), or if it is to be taken as an optional, "all or nothing" count.
- Where Line References are allowed, it can also validate that the line references entered are valid. For a line label like .ABC, the label must exist in your edit file to be "valid". For a line-number pseudo-label like .123, a data line on line 123 must exist, but since these are not actually labels, nothing else need be "defined".
- Where Tag operands are allowed, it can also validate that the Tag operands entered are valid Tags. A tag is valid if at least one tag of that name appears in your edit file.
- Where single keyword operands are specified, it simply tracks whether the keyword was specified or not. It is not an error if the keyword is absent.
- Where mutually exclusive keywords are specified, it validates that one and only one has been entered, This is the same principle used on SPFLite command like FIND, where you cannot say FIND FIRST LAST ABC.
- Where keywords have multiple aliases, it accepts them all and 'normalizes' them to return the first specified keyword. e.g. if the alias list were specified as (EXCL, EXCLUDE, EXC, X) then when any of them are entered it would be treated and returned as EXCL. This normalization process means that you don't have to check every possible spelling of a keyword, but only the first (left-most) one that appears in the list. This greatly simplifies the amount of work needed to check such keywords, since only value has to be tested.

On the completion of a successful **SPF_Parse**, there should normally be no need to examine or use the **Get_Arg\$** array. There will be no unaccounted-for operands 'left over' that would be the responsibility of the macro. (If there **were** unaccounted-for operands, these would be categorized as a syntax error. The fact that you don't **have** a syntax error means the parsing completed successfully).

However, since the initially-created "arg" array is not modified by **Spf_Parse**, nothing prevents access using the functions discussed above, if you have that need.

How to access the parsed-out operands

SPF_Parse sorts and categorizes the command line operands into five groups:

Keywords	<p>For all specified Keywords, you can call a single function to return a TRUE / FALSE indication of whether a particular keyword has been entered or not. For example, to see if the Keyword ALL had been specified, the code would be</p> <pre>IF Get_Arg_KW("ALL") then ...</pre> <p>When keywords are part of a mutually exclusive group (like WORD, PREFIX, SUFFIX) you may call a function to return which of the keywords were specified. The code for this would be</p> <pre>IF Get_Arg_KWGroup\$("list-name") then ...</pre> <p>This requires the creation of list-name to be used to refer to the group of keywords when specifying the keyword list. This is described in the details for SPF_Parse below.</p> <p>Use of the list-name and Get_Arg_KWGroup\$ function does not prevent use of Get_Arg_KW for a specific keyword if you desire.</p> <p>(Just to be clear, all keywords are optional. It would not really make sense to say that a keyword was mandatory every time you used a macro.)</p>
Line References	<p>Your macro may specify that it requires any number of Line References there is no limit. When retrieving Line References, they are returned by relative position, left to right. Their exact operand location, and whether there are intervening operands of other types is immaterial. You request the first Line Reference, the second Line Reference, etc. The code for this would be</p> <pre>start-line = Get_Arg_LRef\$(n)</pre> <p>where 'n' specifies the required relative Line Reference number.</p>
Tag Operands	<p>Your macro may specify that it requires any number of Tag Operands, there is no limit. When retrieving Tag Operands, they are returned by relative position, left to right. Their exact operand location, and whether there are intervening operands of other types is immaterial. You request the first Tag Operand, the second Tag Operand, etc. The code for this would be</p> <pre>tag-name = Get_Arg_Tag\$(n)</pre> <p>where 'n' specifies the required relative Tag Operand number.</p>
Text Literals	<p>Text operands are quoted strings or un-quoted strings which are not keywords nor not solely numeric. i.e. like the search and change strings of a CHANGE command.</p> <p>Your macro may specify that it requires any number of Text Literals, there is no limit. When retrieving Text Literals, they are returned by relative position, left to right. Their exact operand location, and whether there are intervening operands of other types is immaterial. You request the first Text Literal, the second Text literal, etc. The code for this would be</p>

	srch-string = Get_Arg_TextLit\$(n) where 'n' specifies the required relative Text Literal number.
Numeric Literals	<p>Numeric operands are un-quoted strings which are solely numeric digits. i.e. like the column range operands of a FIND command.</p> <p>Your macro may specify that it requires any number of Numeric Literals, there is no limit. When retrieving Numeric Literals, they are returned by relative position, left to right. Their exact operand location, and whether there are intervening operands of other types is immaterial. You request the first Numeric Literal, the second Numeric literal, etc. The code for this would be</p> <pre>start-col = Get_Arg_NumLit\$(n)</pre> <p>where 'n' specifies the required relative Numeric Literal number.</p>

How to specify the macro command syntax

The SPF_Parse function may appear complicated at first, but don't let that hold you back from trying it. Let's have a look:

```
RC = SPF_Parse(TxtLit-number, NumLit-number, LinRef-number,
Tag-number, _
               [ keyword-set, ]
               [ keyword-set, ]
               ...
               )
```

What do we have? Four numeric values, and then a series of optional keyword definitions. Let's see the details.

The first four operands specify how many of each of the non-keyword operands your macro will allow. For example, if your macro operands consists solely of two line reference operands, the coding is simply:

```
RC = SPF_Parse(0, 0, 2, 0)
```

which is pretty straight-forward.

If your macro added an optional "ALL" keyword, the coding becomes:

```
RC = SPF_Parse(0, 0, 2, 0, "ALL")
```

again, not unduly complicated.

How to specify a variable number of values

Many operands are optional. In our example above, the line references may be optional if the macro also allows you to mark the lines with a CC / CC line block. But if we code just the number 2 as the Line Reference operand with nothing else, that means the operands are mandatory, and there must be exactly 2 of them. To allow a varying number of arguments, it is handled by adding the ARG_VAR as a "flag" to the count value, so that this parameter to the SPF_Parse call is changed from 2 to **2+ARG_VAR**. The ARG_VAR option will allow from 0 (zero) to the number specified to be entered.

So if the TxtLit-number operand is coded as **2+ARG_VAR** it means there can be from 0 to 2 Text Literals.

The flag name ARG_VAR, like the ARG_OPT and ARG_DEF flags discussed next, are predefined by SPFLite.

How to specify optional values

Some macro parameters may be used in an optional, "all or nothing" manner. This can be handled by using the ARG_OPT validation flag. When ARG_OPT is added to a count value, it indicates that for this type of parameter the number of supplied values must either be zero or the exact count specified.

For example, if the number of line references is specified as **2+ARG_OPT** it indicates that there must be either **no** line references or **exactly two** line references.

You cannot combine the ARG_OPT flag and the ARG_VAR flag, since the meanings conflict. For instance, if a count value is specified as 2+ARG_VAR+ARG_OPT, the ARG_VAR implies that having one argument is valid, but ARG_OPT implies that it isn't, and these cannot both be true. If you attempt to use the flags this way, the SPF_Parse function will fail with a nonzero RC.

How to validate tag and line references

When Tags or Line References are used as operands, you may wish these entered values to be validated to ensure they correctly point at valid data lines. This can be handled by using the ARG_DEF validation option. When this is added to a value, it indicates that for this type of parameter data line reference must be valid. For example, if the number of line references is entered as **2 + ARG_DEF** it indicates that there must be **two valid** line references.

How to specify keywords

There are two basic types of keywords, the simple "I'm here / I'm not here" type of keyword (like the RAW operand of CUT), and the mutually exclusive list of keywords (like ON / OFF, or WORD / PREFIX / SUFFIX)

For the simple type, the keyword-set operand is just the keyword itself, couldn't be simpler. "ALL" or "TRUNC" etc.

For the mutually exclusive type, the various keywords are just entered, separated by commas. e.g. for an ON/OFF pair, the keyword-set would be coded as "ON, OFF". For the WORD / PREFIX / SUFFIX example it would be "WORD, PREFIX, SUFFIX"

When a keyword can be specified in various alias values, the normal single keyword value in the list is replaced by the list of aliases, enclosed in parentheses. As an example, a macro might accept either DELETE or PURGE as operands. These would normally be coded as "DELETE, PURGE". But what if you wanted to accept DEL as an alternate for DELETE and PUR as an alternate for PURGE. It would then look like "(DEL, DELETE), (PUR, PURGE)"

Almost done now, just one more wrinkle for keywords. In the above examples, any of the keywords can be tested for using the Get_Arg_KW function, but what if, for the WORD/PREFIX/SUFFIX example, you wanted to ask "Which one was entered" without having to test each one individually? This is possible, but you have to assign a 'list-name' to refer to that specific set of keywords. Then you can ask for which KW in the set was entered by using the Get_Arg_KWGroup\$(list-name).

So where does the list-name go? It is inserted at the beginning of the list, followed by a colon (:). Our example here, using the list-name WTYPE, would be coded as:


```
"WTYPE:WORD,PREFIX,SUFFIX".
```

Asking for 'which one was entered' would be done by

```
wtype = Get_Arg_KWGroup$("WTYPE")
```

Example: Putting SPF_Parse all together

A picture is worth a thousand words, so lets do an example. Here's a reasonably complex imaginary macro command whose syntax is similar to FIND and looks like:

```
MACNAME search-str
[ start-col [ end-col ] ]
[ start-line [ end-line ] ]
[ WORD | PREFIX | SUFFIX ]
[ RED | GREEN | BLUE | YELLOW ]
[ ALL ]
```

Lets code the SPF_Parse request to parse this command:

```
IF SPF_Parse(1, 2 + ARG_VAR, 2 + ARG_VAR + ARG_DEF, 0, _
             "WTYPE:WORD,PREFIX,SUFFIX",
             "COLOR:RED,GREEN,BLUE,YELLOW", _
             "ALL") then
    Halt(FAIL, Get_Msg$)
END IF
```

- The first operand **1** says there is 1 mandatory text literal (the search-str in the syntax diagram)
- The second operand **2 + ARG_VAR** says there may be 0 to 2 optional numeric literals (the start-col and end-col in the syntax diagram)
- The third operand **2 + ARG_VAR + ARG_DEF** says there are from 0 to 2 optional line references (the start-line and end-line in the syntax diagram and that if entered, the references should be validated as correct line references)
- The fourth operand **0** says there are no allowable Tag operands.
- The **"WTYPE:WORD,PREFIX,SUFFIX"** operand specifies 3 mutually exclusive keywords, which can be referenced as a group using the list-name value of **WTYPE**.
- The **"COLOR:RED,GREEN,BLUE,YELLOW"** operand specifies 4 mutually exclusive keywords, which can be referenced as a group using the list-name value of **COLOR**.
- The **"ALL"** operand specifies the simple keyword **ALL** which may be present or not present.
- The entire **SPF_Parse** function is coded as part of an **IF ... THEN** statement. Since **SPF_Parse** returns a non-zero value (8) if any parsing errors occur, it is usually simplest to test this way and use an **SPF_SetMsg** to issue the error message text provided by **SPF_Parse**. This text is obtained using a standard **Get_Msg\$** function.
- Assuming **SPF_Parse** returns with no error status, the macro can continue at this point to use the parsed values, or to perhaps perform additional validation if required by the circumstances of the macro's function.

A simple macro demo in all three command operand modes

Here is a simple macro coded in the three different modes to show the differences. The syntax for this example is:

```
DEMOFIND srch-string [ ALL ]
```

Note: All examples below should probably use `SPF_Quote$` to properly process the `srch-string`, but this has been left out to simplify the demonstration code.

Basic Operand Access

```
' DEMOFIND.MACRO
DIM OpAll as STRING
IF Get_Arg_Count < 1 then Halt(FAIL, "Missing search
string")
OpAll = ucase$(Get_Arg$(2))
If OpAll <> "" and OpAll <> "ALL" then Halt(FAIL, "Unknown
operand: " + OpAll)
SPF_CMD("FIND " + Get_Arg$(1) + " " + OpAll)
HALT
```

SUB Operand Access

```
' DEMOFIND.MACRO
SUB DEMOFIND(srchstr as string, OpAll as STRING)
IF srchstr = "" then Halt(FAIL, "Missing search string")
If OpAll <> "" and OpAll <> "ALL" then Halt(FAIL, "Unknown
operand: " + OpAll)
SPF_CMD("FIND " + srchstr + " " + OpAll)
Halt
END SUB
```

Full Parse Access

```
' DEMOFIND.MACRO
if SPF_Parse(1, 0, 0, 0, "ALL") then Halt(FAIL, Get_Msg$)
SPF_CMD("FIND " + Get_Arg_TextLit$(1) + iif$(Get_KW("ALL"),
" ALL", ""))
HALT
```

Macros for fun and profit

A brief overview of macros

Macros for fun and Profit

We have (somewhat whimsically) entitled this section, *Macros for fun and profit*. You might be persuaded that you could profit from using macros, but maybe you think you won't have much fun writing them. Let's see what we can do to change your mind !

Users sometimes feel that the whole topic of macros is too complicated, and they tend to steer clear of them, without even trying to see what they could accomplish by taking advantage of their capabilities.

Are SPFLite macros really that complicated? Well, you could certainly **make** them complicated if you were inclined to - but then that would be **your** doing, and not SPFLite's fault!

There's actually not that much that is inherently "hard" about these macros, once you familiarize yourself with the **thinBasic** syntax and features. It's really the same as with the rest of SPFLite itself, which has a large array of primary and line commands and keyboard functions at your disposal. That doesn't necessarily make it "hard". It just means you have a lot of tools that you **could** use if you needed them. The specific commands you use, and how you use them, are driven by your requirements, which in turn are driven by the nature of the data you are creating or modifying.

The same is true with macros. A given macro is really only as complicated as the editing task you are trying to carry out. A trivial task can be done with a trivial macro, but a very complex task calls for an involved macro.

That's only fair. If you wrote some external script in Perl, Rexx or C++ to do a complex editing process, **that** would be involved, too. This is computation - not magic. You can't get something for nothing.

So, the main point is: *don't panic*. Most SPFLite users that create macros will probably only need relatively short scripts. But, the tools and features are there to do really cool and complicated things - **if** that's what you need and want to do.

Let's try and show you how straightforward a macro can be. The best way to do that is with an example, so let's choose a simple one that you should find easy to understand.

Example: The ONLY macro

The ONLY macro implements a two-step SPFLite command sequence:

```
EXCLUDE ALL
FIND ALL string
```

This changes the display to show ONLY those text lines containing a specified *string*.

In the "good old days" of SPFLite 1.0, as well as in other editors like ISPF, early Tritus SPF, and SPF/SE, this is the only way you **could** do that. SPFLite presently allows you perform this

function directly using the built-in command NEXCLUDE. But, bear with us - imagine that SPFLite **didn't** have this command built in, but you needed to do it anyway. After all, eventually you **will** find some task you need to do that SPFLite doesn't do directly, and then you really will need a macro.

Invoking SPFLite commands within a macro is done by calling a function designed for this purpose, the **SPF_Cmd** function, which is used like this:

```
SPF_Cmd("some SPFLite primary command")
```

The command you specify can be any valid thinBasic string expression, but for now we'll start with simple string literals.

By the way, in the example above, "some SPFLite**primary** command" means that you can't (directly) enter **line** commands using this function. However, you **can** use the **LINE** primary command to achieve the same thing. See the [LINE](#) primary command in the main Help documentation for more information.

So, what we want to create is a macro script containing the lines

```
SPF_Cmd("EXCLUDE ALL")  
SPF_Cmd("FIND ALL XXX")
```

Let's do that, and create the file **ONLY.MACRO** in the SPFLite \MACROS folder.

FYI, if you're going to get in the business of creating and editing macros frequently, you may find it convenient to set up a FILELIST so you can see all of your macros in one place in the File Manager. See the FILELIST documentation in the main Help file for more information, but basically, simply create a file named MACROS.FILELIST containing the following line. Store in the SPFLite data directory.

```
C:\Users\username\My Documents\SPFLite\MACROS|*.*
```

However, SPFLite has one additional requirement when creating a macro. The first line of it must be a macro prototype (or, *header*) statement, which takes the form of a thinBasic comment line in a special format. In our example, it requires little more than the name, and our macro now looks like this:

```
' ONLY.MACRO  
SPF_Cmd("EXCLUDE ALL")  
SPF_Cmd("FIND ALL XXX")
```

That wasn't so bad, was it?

Our macro is complete ... except that right now, it always does a FIND command for a literal string of "XXX". That "XXX" was just a place-holder, while we were busy trying to explain things and setting up the macro. But, we originally wanted to do a FIND operation that looked like **FIND ALL *string***, where *string* is supplied on the primary command line - remember? Somehow, we need to create this command dynamically, so it includes **your** string, and not just XXX all the time.

To do that, we need to have the SPF_Cmd function accept a *string expression*, rather than a *string literal*. So, let's make a simple change to our macro to do just that:

```
' ONLY.MACRO
```

```
SPF_Cmd("EXCLUDE ALL")
SPF_Cmd("FIND ALL " + Get_Arg$(1))
```

Here, the string literal "FIND ALL " and the string returned by the function `Get_Arg$(1)` are concatenated together to form a single string value. The `Get_Arg$(n)` function returns the string contents of the specified argument number. The + plus sign is the string concatenation operator (you can also use & ampersand if you like; they both mean the same thing).

Remember to put a blank after the "FIND ALL " string, like we did here. Otherwise, the ALL and the string you got back from `Get_Arg$(1)` would get "run together". If you did that, a macro command like ONLY ABC would cause the macro to create a command string expression like **FIND ALLABC** which would either be illegal, or just not what you wanted.

So now, if the command ONLY FRED were issued, `Get_Arg$(1)` returns the string "FRED", and so the two commands issued would effectively become:

```
EXCLUDE ALL
FIND ALL FRED
```

Exactly what we want!

But, suppose we want to include optional FIND operands like WORD or PREFIX or LAST. This is another simple change, and our final macro will now look like:

```
' ONLY.MACRO
SPF_Cmd("EXCLUDE ALL")
SPF_Cmd("FIND ALL " + Get_Arg$(0))
```

Hmm, what's different? The `Get_Arg$(0)` instead of `Get_Arg$(1)` is a request for **all** the operands, not just the first. (When you provide two or more macro operands to ONLY, the `Get_Arg$(0)` function returns all of them together, so they are separated from each other by a blank.)

So, if the macro command ONLY FRED WORD were issued, the two commands issued would be:

```
EXCLUDE ALL
FIND ALL FRED WORD
```

Not too shabby, eh?

Enhancing the ONLY macro to work in more than one context

Certainly, that was a fairly simple example, but like most macros, it grows once you start using it and think "It would be nice if ...".

So, what more could we possibly do with this ONLY macro? Well, a lot of the time the ONLY macro would be issued while you were browsing a file and you saw some word of interest within the text. Why should we have to manually type in (or even cut and paste) a word on to the command line, when it's already sitting right there in the text? (There's something to be said for constructive laziness! - RH) Let's change ONLY so that it will use the word that's being pointed to by the cursor.

This is also surprisingly easy. Our macro now becomes:

```
' ONLY.MACRO
SPF_Cmd("EXCLUDE ALL")
SPF_Cmd("FIND ALL " + Get_Curr_Word$)
```

The `Get_Curr_Word$` function is designed specifically to address this need. It returns the 'word' that the cursor is sitting on, when the cursor is underneath some string in the data area of the edit window. Now, we can map a keyboard key (let's say, Ctrl-O) to ONLY and then all we need to do is place the cursor anywhere within a word and hit Ctrl-O.

We now have a quite useful macro, and it is still only 3 lines long.

However, we have now lost the ability to actually type in an **ONLY string** command, since the macro now requires a word to be present under the cursor. If you are typing a command on the **command** line, there isn't any file data under the cursor, because your cursor is not **in** the data area.

But, we don't want to give up the command-line usage. What we really need is a way to determine the macro's **context** - that is, **how and where the macro is being used**. Then, we tailor the macro to operate differently depending on how it's being used, and then we can use the same macro for more than one purpose.

Managing quoted operands

Before we put this new macro together, there is a little matter of quotes to deal with. If you recall from the basics of SPFLite, you can have quoted string values enclosed in ' single quotes, " double quotes or ` accent quotes, or simple strings can be supplied unquoted.

You also need to be aware that keywords that are reserved elsewhere, like ALL and LAST, are **not** reserved as operands of a macro used as a primary command.

So, if you supply a macro argument that happens to be a keyword of some other SPFLite command, and you then use that string to dynamically create some command, it could create some invalid syntax. For example, without some "fancy footwork", if we tried to say ONLY FIRST, one of the created commands would be FIND ALL FIRST, and that's not legal.

In your macro, you might be tempted to get around this by always putting quotes around the string you get back from a `Get_ARG$` call. But, suppose, as a macro user, you actually placed quotes around a string, like ONLY "FIRST". If your macro added quotes, too, you'd have too many of them. So, if you issued a macro command like ONLY "FIRST" with quotes already around the operand, and the macro logic added quotes, too, you'd end up with a command like FIND ALL ""FIRST"" with improperly-doubled quotes - and that's not what you wanted.

You **could** check to see if the argument already had quotes on it, and tailor your expression accordingly, but that would be a real nuisance to do all the time. Fortunately, you don't have to. We have created a function called `SPF_Quote$`. What this does is it adds quotes to a string expression, unless it is already quoted, in which case it leaves it as is. It also takes into account whether a string contains inner quotes of any type, so that it wouldn't create a wrongly-quoted string. It does that by choosing one of the three allowable quote types, to avoid having the enclosing quotes be the same as any inner quotes which are literal data values. These actions make the function what we call a "smart quoter".

The example below uses the smart-quoter function to correctly create SPFLite command strings. Now, even if the operands you provide are already quoted or are SPFLite reserved words, the macro will still work properly.

You would need this smart quoter function whether the operand was supplied on the command line or was obtained from a function like `Get_Curr_Word$`, because we have no way of knowing if the word under the cursor is reserved either.

Managing argument lists

When you use the ONLY macro, and it has multiple arguments, you can "grab" them all with a function. But if you needed to quote the search string AND you had additional arguments, how could you just quote the first one and not the rest of them?

This is going to take a little tinkering ... but we promise, it won't be too hard.

First of all, you need to understand that macro parameter lists are **not** like SPFLite command parameters. For example, a FIND command could say FIND ALL ABC or it could say FIND ABC ALL, and they both mean the same thing, because you can supply operands to most SPFLite commands in any order. SPFLite can tell which operands are reserved words and which are user-supplied values, and it sorts them out based on which command you used.

Although Macros can be coded to handle operands this way, for the purposes of this discussion we will ignore that temporarily and keep things simple. If, following this discussion, you want to learn how to handle operands in that manner, review [Full Parse Access](#).

With that in mind, we will agree to design the ONLY macro so that the *string* operand is always the first one, meaning that you'd always use `Get_Arg$(1)` to "grab" it. What about all the *other* operands - if there are any? How do we grab them?

First, we take advantage of the special fact that `Get_Arg$` can be called with either one or two arguments of its own. If called with two arguments, you provide a "range" of operands. So, if you wanted arguments 3 through 5, you could get them with `Get_Arg$(3,5)`.

You have to ask for arguments in ascending order. Calling `Get_Arg$(5,3)` will return nothing (an empty string). And also trigger a failing RC value and associated error message if you cared to check and retrieve them.

Second, if you read closely in the Function Details section, you'll find there is a function called `Get_Arg_Count`. So, assuming you wanted all arguments starting with the second one, you could say `Get_Arg$(2,Get_Arg_Count)`. That would work fine, but it's kind of wordy. Since it will be a common task to require "all remaining operands" this way, the function allows the second operand to be specified as 0, with a call like `Get_Arg$(2,0)`. When you supply a 0, it is treated the same as if you had used `Get_Arg_Count` instead. That makes this a lot shorter and easier to type.

A call like `Get_Arg$(2,0)` is not considered breaking the rule about asking for arguments in ascending order; it's just a short-cut convenience so you don't have to worry so much about exactly how many arguments were present, and so you don't have to type so much.

Finally, if a call to `Get_Arg$` asks for more arguments than you supplied when you ran the macro, it will only provide the ones that are there. It's not illegal to ask for "too many".

Just like the single-argument form `Get_Arg$(0)`, when you provide two or more macro operands and use `Get_Arg$(2,0)` it returns the requested macro operands so they are separated from each other by a blank.

Putting it all together

So, let's finish the job on our macro and flesh it out, adding a bit of error checking as well.

Note that `Get_Arg$` will put spaces *between* the operands it returns, but not before or after. So, in the `SPF_Cmd` expressions, we have to separate the various "pieces" of the expression with a blank so that we end up creating a valid SPFLite primary command; otherwise macro argument 1 and 2 would get "run together" and things wouldn't work right.

Notice too that we only quote the first argument, but not the remaining ones. Why not? Because we agreed that the "string" argument would always be the first one (remember our discussion from above?) which means the remaining arguments, if there are any, would only be SPFLite keywords like `WORD` or `LAST`, which would never be quoted if you wanted to use them as normal keywords and not as data. So, we don't call `SPF_Quote$` for them.

```
' ONLY.MACRO
if Get_Arg$(0) <> "" then
    SPF_Cmd("EXCLUDE ALL")
    SPF_Cmd("FIND ALL " + SPF_Quote$(Get_Arg$(1) + " " +
Get_Arg$(2,0)))
else
    if Get_Curr_Word$ <> "" then
        SPF_Cmd("EXCLUDE ALL")
        SPF_Cmd("FIND ALL " + SPF_Quote$(Get_Curr_Word$))
    else
        Halt(FAIL, "There is no word under the cursor")
    end if
end if
```

Several changes have been made:

- If a command line argument is present, then **that** is what is searched for
- If no command line argument **and** the cursor is on a word, **that** word is searched for
- If no command line argument and the cursor is not on a word, an error message is issued using the `Set_Msg` function
- The smart-quoter function `SPF_Quote$` ensures that if operand 1 is quoted or is a reserved word, it won't cause an incorrect `FIND` command to be created.

Our final, fully fleshed-out macro is still only 12 lines, including the header. That's pretty reasonable, and hopefully not too intimidating, even if you're new to writing macros.

Simplifying the ONLY macro with NEXCLUDE

If you look closely, by providing some extra logic to take the context into consideration, you now have a capability that SPFLite doesn't have on its own - even the most current version. Remember we said the basic functionality of `ONLY` is already covered by the built-in command `NEXCLUDE`? Well, what `ONLY` does **now** is actually **more** powerful than `NEXCLUDE`. But, that doesn't prevent you from actually using `NEXCLUDE` here. We can eliminate two lines from the macro by replacing `EXCLUDE` and `FIND ALL` with `NEXCLUDE ALL`.

This is a case where having a good familiarity with all of SPFLite's array of available commands can help you write macros that are shorter and run faster. Here is how the revised `ONLY` macro would now look:


```

' ONLY.MACRO
'
'
if Get_Arg$(0) <> "" then
    SPF_Cmd("NEXCLUDE ALL " + SPF_Quote$(Get_Arg$(1)) + " " +
Get_Arg$(2,0))
else
    if Get_Curr_Word$ <> "" then
        SPF_Cmd("NEXCLUDE ALL " + SPF_Quote$(Get_Curr_Word$))
    else
        Halt(FAIL, "There is no word under the cursor")
    end if
end if
end if

```

Extra credit - Part 1

Did you notice how we determined the 'context' of the macro - how it was being used? Calling **Get_Arg\$(0)** returns a string with all the arguments provided to the macro. If none were provided, you'd get an empty string.

Suppose your macro is called **AX.MACRO**. Under what circumstances would you get an empty string back from **Get_Arg\$(0)** when you used this macro?

- You enter **AX** on the command line by itself with nothing else, to run the macro as a primary command, and press Enter.
- You enter **AX** or **AXX** in the sequence number area of one or two lines, and press Enter. (If you use **AX** as a line-command macro, only one may be specified at a time. If you use **AXX** you must specify it in pairs. Otherwise, SPFLite will detect a usage error.)

Is that the **only** way to determine the context? No, there are a few other ways, too:

- You can call **Get_Arg_Count** to find out how many arguments were present; if there weren't any, you get 0
- You can call **Get_Csr_LPTR** to find out the line-pointer where the cursor is located; if it's not in the main edit area, you'd get a 0
- You can call **Get_LNUM(Get_Csr_LPTR)** to find out the line-number where the cursor is located; if it's not in the main edit area **or** it's on a non-data line like BNDS or TABS, you'd get a **0** (since lines like BNDS and TABS **have** no line number). This is a little more involved test, but it's a "stronger" one.

You can also see whether your macro was called as a primary-command macro, or as a line-command macro. See [Primary mode vs. Line mode macros](#) for more information.

Based on your requirements, you'd have to decide how "strict" your test needs to be to ensure it only runs under the right circumstances.

Extra credit - Part 2

Now that we have created the ONLY macro, fine-tuned its behavior, and taken its operating context into account, what do you think:

Is ONLY a primary-command macro or a line-

command macro?

Yes, it's a trick question - and the answer is, surprisingly, **both**.

The ONLY macro works quite well when invoked as a line-command macro. If you enter ONLY in the sequence area of a data line, then move the cursor over the word you want as the operand, and press Enter, the macro will function correctly. In fact, the ONLY macro name could be in the sequence area of one line, and the cursor could be on an entirely different line, and it would still work.

What happens? First, the Get_Arg\$(0) call will return an empty string, because the macro has no arguments, per se. It then finds a value returned from the call to Get_Curr_Word\$, and the macro logic proceeds from that point.

No, this isn't likely to be a convenient way to type this, and most users would not do it that way, but unless a macro is carefully written to insist on being run only one way or the other, it may very well meet all the requirements for execution as either a primary-command macro or a line-command macro - even if that isn't what you intended when you wrote it.

Most of the time, you will write a macro with the intent of it only running one way, either as a primary-command macro or a line-command macro, and you may neither know nor care that it would work the 'other' way.

If you need to be sure, you can use the [Is Line Cmd](#) and [Is Primary Cmd](#) functions. See [Primary mode vs. Line mode macros](#) for more information.

Working with Global Data

Introduction

The SPFLite macro language provides for the storage and retrieval of numeric and string values in a **Global** manner. That is, the data is stored in a single data area associated with a single SPFLite Instance. The data is shared by all active Edit tabs (and the FM tab) and is only freed when the whole SPFLite window is terminated.

Thus, a macro (or multiple co-operating macros) can pass data between separate Edit tabs and/or separate invocations of the macros. Think of it as a sort of 'scratch-pad' of data, available throughout all SPFLite macros.

Storage Structure

Tables

Global data is saved in tables which are identified by a table number. This table number is optional. If no table number is specified, SPFLite will use table number 0.

You may use any numbers you choose, they do not have to be consecutive, and all table numbers are treated equally.

There is no limit to the number of tables, the range of numbers used, or the total amount of data stored (well, limited by available memory)

Entries

Each entry in a table consists of a 'Name' and 'Value' pair. Entries are organised according to the data type of the 'value' being stored.

Use the **...Gbl_Num...** functions to manipulate numeric 'value' entries and the **...Gbl_Str...** functions to manipulate string 'value' entries.

There are two complete sets of functions, whose only difference is the type of data being stored.

The available functions are:

Numeric values	String Values
Set_Gbl_Num	Set_Gbl_Str
Get_Gbl_Num	Get_Gbl_Str\$
Get_Gbl_Num_Count	Get_Gbl_Str_Count
Get_Gbl_Num_Name\$	Get_Gbl_Str_Name\$
Get_Gbl_Num_TableName\$	Get_Gbl_Str_TableName\$
Delete_Gbl_Num	Delete_Gbl_Str

Storing and Retrieving data in Global Storage

Storing

Set_Gbl_Num([Table-Num,],Name,Value) is used to create/replace a numeric 'value' entry

Set_Gbl_Str([Table-Num,],Name,Value) is used to create/replace a string 'value' entry

Both functions accept the following operands:

Table-Num	Optional	If specified, it must be the first operand. If omitted, table number 0 is assumed.
Name	Required String	This is a Name to identify the 'value' to be stored.
Value	Required String/Number	This is actual 'value' to be associated with the preceding Name.

Retrieving

variable = Get_Gbl_Num([Table-Num,],Name) is used to retrieve a numeric 'value' entry

variable = Get_Gbl_Str([Table-Num,],Name) is used to retrieve a string 'value' entry

Both functions accept the following operands:

variable	Required	The variable to receive the data value. The function can also be used directly as a macro numeric operand.
Table-Num	Optional	If specified, it must be the first operand. If omitted, table number 0 is assumed.
Name	Required String	This is a Name to identify the 'value' to be retrieved.

Removing / Deleting Table entries

Deleting an Individual Table Item

Delete_Gbl_Num([Table-Num,],Name) is used to delete a single numeric 'value' entry. i.e. a 'name' and 'value' pair.

Delete_Gbl_Str([Table-Num,],Name) is used to delete a single string 'value' entry. i.e. a 'name' and 'value' pair.

Both functions accept the following operands:

Table-Num	Optional	If specified, it must be the first operand. If omitted, table number 0 is assumed.
Name	Required String	This is a Name to identify the 'value' to be deleted.

Deleting an entire Table's Contents

Reset_Gbl_Num([Table-Num]) is used to delete all numeric entries. i.e. 'name' and 'value' pairs from the specified table.

Reset_Gbl_Str([Table-Num]) is used to delete all string entries. i.e. 'name' and 'value' pairs from the specified table.

Both functions accept the following operand:

Table-Num	Optional	The table number to be cleared. If NO Tbl-Num operand is entered, ALL tables are cleared. Entering a 0 (Zero) is NOT the same as omitting the operand, it will simply clear table 0.
-----------	----------	--

Retrieving/Scanning All Data in a Table

Usually Gbl data is accessed directly using the table-number and the name. Retrieval of all items in a table is also possible. It is not, however, as straightforward as it might be. Accessing items in this way requires additional effort in the macro in order to enumerate the *table-number* and *name* fields with which to retrieve the stored information. Once these are known, data can be accessed using the normal Get_Gbl_xxx functions.

The following three additional functions permit such enumeration.

Get_Gbl_xxx_Count([Table-Num]) is used to fetch the current item count in the table

Get_Gbl_xxx_Name\$(Index-Num) is used to fetch the *Name* for the entry specified by *Index-Num*.

Get_Gbl_xxx_TableName\$(Index-Num) is used to fetch the *table-num* and *Name* for the entry specified by *Index-Num*.

Get_Gbl_xxx_Count([Table-num])

This returns the count of items in the table. If *Table-Num* is omitted, the count is for **all** items in **all** tables. If *Table-Num* is provided, the count is for **that** table.

Get_Gbl_xxx_Name\$(Index-Num)

This will normally be used in a loop, from 1 to the # returned by **Get_Gbl_xxx_Count**. It will return the *name* associated with that specific Index-Num. The returned name can then be used in a **Get_Gbl_xxx** function to retrieve the actual data value.

Note: This returns the *name* **regardless** of the table in which it is found. If the table number is significant, use the next function (**Get_Gbl_TableName\$**).

Get_Gbl_xxx_TableName\$(Index-Num)

This will normally be used in a loop, from 1 to the # returned by **Get_Gbl_xxx_Count**. It will return a string of format **Table-Num,name** associated with that specific Index-Num. With the returned string in *str-var*, the table number is easily accessed via *VAL(str-var)*. Similarly, the Name value can be accessed via *REMAIN\$(str-var, ",")*. The returned value can be easily parsed into *Table-Num* and *label-str*.

Note: This function provides the *table-num* and the *name* which allows you to select only items in a specific *table-num*.

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

SPFLite Interface

The **thinBasic** script interpreter itself has no knowledge of SPFLite whatsoever. It knows and understands Basic syntax, and nothing else.

But, when run from within SPFLite, a collection of SPFLite interface routines are automatically made available as extensions to the **thinBasic** language. These routines provide all the needed tools to support interaction with SPFLite and allow **thinBasic** to be used as a proper scripting language. These functions provide the following abilities:

- Functions which can access a variety of status information about the SPFLite edit session itself and the file being edited.
- In FM mode, functions to fetch info about the current FM display. e.g. the current display criteria.
- Functions which can be used to:
 - Fetch and store data lines in the Edit session
 - Issue Primary and Line commands that affect the Edit session
 - Set macro return codes and messages for the user
 - Set cursor location to be used following the macro execution
 - Interface with File Manager to process the displayed files

Note: You may notice that one of the features of SPFLite that is **not** supported through macros is the interactive invocation of keyboard primitive functions.

The introduction of the *SPF_Post_DO* function, which can be used to establish a DO macro which will be run following normal macro termination can, in some cases, overcome the restriction that macros cannot invoke activities involving multiple tabs.

Interface Structure

Macro Interface Structure

[Passing control to your macro](#)

[Primary mode vs. Line mode macros](#)

[Passing information between macros and SPFLite](#)

[Identifying and referencing lines](#)

[Understanding Line Pointers and Line Numbers](#)

[Line Pointers or Line Numbers: Which is Better?](#)

[The SPF_CMD function and Line Numbers vs. Line Pointers](#)

[Understanding Source and Destination Line Ranges](#)

[File Manager Interfacing](#)

Passing control to your macro

SPFLite determines that a command **is** a macro when a "command-name.MACRO" file is found in the SPFLite \MACROS folder. A macro name **can** be the same as a built-in primary command, and will replace the built-in command. If the macro wishes to pass control to the built-in command, it may do so with a simple SPF_CMD(command-string) as internally issued commands are **never** treated as macros.

It **is possible** to make a macro used as a **primary** command with the same name as built-in **line** command. For example, a primary-command macro named CC.MACRO is valid. However, that could be confusing, and for most people, it will be best if you avoid **any** built-in command names, whether line-mode or command mode. This is especially true in case you write a macro that could be used both as a primary-command macro or as a line-command macro.

When SPFLite detects a macro is being invoked, it triggers the **thinBasic** interpreter and passes it two things: the source of your macro from the macroname.MACRO file and the list of available SPFLite interface functions being made available.

The number of available interface functions is extensive (well over 150) but don't let this put you off. Just like SPFLite's extensive list of Primary commands, Line commands, and Keyboard primitives, nobody will ever need or use all of them. We're just trying to ensure that we provide as extensive an interface as possible. Nothing is worse than developing a macro and saying "Gee, if there was only a function that would". We don't want that to happen to you.

thinBasic parses and analyzes your source code and begins execution of the interpreted code. At this point, the macro source code has still been provided no information whatsoever about SPFLite, or the file being edited.

All information passed between the macro and SPFLite is handled via those custom functions which SPFLite made available when invoking **thinBasic**.

If you're a compiler geek or something, what SPFLite does is "inject" the SPFLite interface functions into the **thinBasic** namespace, so that you don't have to declare or "include" them. (If you're not a compiler geek, then it's magic - and don't nobody ask no more questions.)

Primary mode vs. Line mode macros

SPFLite allows you to write macros that can be used on the primary command line, or in the line command area. The macro has the same structure either way. You would have to determine how your macro is getting called, by making queries against some SPFLite built-in functions. Similarly, macros can be written for use in normal Edit sessions and also to execute in the File Manager tab.

NOTE: You cannot have an SPFLite interaction in which you have macros used as line commands, **and** any command on the primary command line (whether built-in SPFLite primary command or a primary-command macro), at the same time.

When you use a macro as primary command, and you have any pending line commands at the same time, these would usually consist of an **A** or **B** and/or a **CC** block, to define the source lines and optional destination line.

When a macro is being used as a primary command

- [*Is FM*](#) will return TRUE if running in the File Manager tab, otherwise FALSE
- [*Is Primary Cmd*](#) will return TRUE
- [*Is Line Cmd*](#) will return FALSE
- [*Get Src1 LPTR*](#) and [*Get Src2 LPTR*](#) return non-zero values if you have a line range defined using a **CC** or **MM** block, otherwise these will return zero.
- [*Get Src LCMD\\$*](#) will return a **C/CC** or **M/MM** etc. if a line command is present, otherwise an empty (null) string value
- [*Get Dest1 LPTR*](#), [*GET Dest2 LPTR*](#) and [*Get Dest LCMD\\$*](#) will be set to appropriate values if you have defined a destination line range using **A**, **AA**, **B**, **BB**, **H**, **HH**, **W**, **WW**, **O**, **OO**, **OR** or **ORR** line commands, otherwise these will be zero or null.

When a macro is being used as a line command

- [*Is FM*](#) will return TRUE if running in the File Manager tab, otherwise FALSE
- [*Is Line Cmd*](#) will return TRUE
- [*Is Primary Cmd*](#) will return FALSE
- [*Get Src1 LPTR*](#) and [*Get Src2 LPTR*](#) will return non-zero values
- [*Get Src LCMD\\$*](#) will return the contents of the line command used to invoke the macro. For example, a macro **AX** is invoked in response to a line command of **AX** or **AXX**.
- [*Get Dest1 LPTR*](#), [*GET Dest2 LPTR*](#) and [*Get Dest LCMD\\$*](#) will be set to appropriate values if you have defined a destination line range using **A**, **AA**, **B**, **BB**, **H**, **HH**, **W**, **WW**, **O**, **OO**, **OR** or **ORR** line commands in addition to the macro defined source range, otherwise these will be zero or null.

The name of a line-command macro is the name you'd use for a single line. So, **for example**, if you wanted to create a line command called **AX**, your macro would be called **AX.MACRO**. If

you want to use this command as a block-mode line-command macro, you can do it any of the normal ways you'd expect with an SPFLite built-in line command:

- Define the beginning and end of the block with **AXX**
- Put **AX** on a line, and follow it with an **n**, **/** or **** modifier
- Create a **SET** name of the form **SET MACROMODE.AX = BLOCK** if you only want to use **AX** as a block-mode command

Passing information between macros and SPFLite

To accomplish its desired actions, your macro must use the provided SPFLite functions to both fetch and to store data back and forth between SPFLite's data areas and the macro's own variable areas.

There are four main types of provided functions:

Get_XXXX
FMGet_XXXX The **Get_XXXX** and **FMGet_XXX** function calls will return the value of the requested SPFLite data item to the macro. These function calls can effectively be used as if they are variables. **Get_XXXX** functions ending in a \$ sign (like **Get_PATH\$**) return character strings, those without a \$ sign return numeric values. (This is a standard BASIC naming convention.)

Set_XXXX
FMSet_XXXX The **Set_XXXX** and **FMSet_XXXX** function calls are used to alter the value of a specified SPFLite data item using the provided macro value.

Is_XXXX The **IS_XXXX** function calls are used to test the basic characteristics of the environment, of data items, and to test the line type of a specified line within the Edit session. SPFLite provides access to all of the different line types (including "special" ones), not just text data lines. Access is provided to NOTE lines, MASK lines, etc. The **Is_XXXX** function calls will return TRUE if the line **is** the requested type, and FALSE if it is any other type. For example, to test if a specific line is a normal text data line you would code **IF Is_DATA(line-ptr) THEN**

Because special lines are not data lines, they only can have a line pointer to them - but never a line number; whereas a data line can have both a line pointer and a line number.

SPF_XXXX The **SPF_XXXX** function calls are used to request specific SPFLite services of a more complex nature, or ones which cannot be strictly described as a GET or SET operation. **SPF_CMD** for example is used to pass a primary command to SPFLite for execution.

Each of these function calls is described in the [Function Overview](#) and [Function Details](#) sections.

Identifying and referencing lines in an Edit Session

To inspect or modify the lines in an Edit file, you must be able to identify which line of the file

you are referring to when using the supplied SPFLite interface functions. This seems simple enough - couldn't we just use the visible line number itself to reference a line? **No.**

We need to be able to access all the different line types within an edit session, not just the text data lines. An edit session may have "special" lines such as **NOTE**, **BNDS**, **COLS**, **TABS**, etc. that we might want to access, and these lines don't have line numbers.

The solution is to use something called a **line pointer** as the key to accessing the data. As long as you keep in mind that a line pointer can sometimes point to non-text lines, most of the time you can think of it **as** a line number. To simplify things, most (but not all) of the custom interface functions work with line pointer parameters. We have been careful with naming conventions so you'll know what functions take what kind of arguments, and there are ways of converting between line numbers and line pointers.

The difference between a line **number** and a line **pointer** becomes an issue when calling the `SPF_CMD` function, and when interfacing with the external user (yourself) since the user never sees and is not aware of the line pointer values. So, line number values are provided by the user, and you may want to pass back line number values to the user, for example, in messages. To do this, you will need the conversion functions.

Conversion between a line **number** and a line **pointer** is a simple process. There are two companion functions whose sole purpose is to convert line pointers to line numbers, and line numbers to line pointers. A side benefit of the line number to line pointer conversion function is that it incorporates automatic support for line labels as well as line numbers. In that sense, it doesn't just accept **line numbers** but more generic **line references**, of which line numbers are one type.

The two conversion functions are [`Get_LNum`](#) and [`Get_LPtr`](#).

But the most powerful reason for using line **pointers** over line **numbers** is that line **pointers** are amenable to being adjusted arithmetically while line **numbers** are not. Why so?

Line numbers, cannot be guaranteed to always increment by 1.

Line pointers however, will always point at a valid line of some type. (lets ignore the Top and Bottom boundary conditions for now). So you can always do a line-pointer + 1 calculation and you will always end up with a valid line pointer. This is the main reason that the macro interface functions are set up to use line **pointers** almost exclusively. It's just easier in nearly all respects.

Understanding Line Pointers and Line Numbers in an Edit Session

A line **number** is the numeric value of the sequence number you see displayed on a data line. So, if you see a data line with a sequence number **000123** then the line number is **123**. A valid line number will never be zero.

A line **pointer** may be thought of as a numeric index to every visible line (special or ordinary) of the entire edit file, starting with 1 for the Top of Data line, and incremented by 1 for each subsequent line (special or ordinary) until the Bottom of Data line is reached.

That is an important fact to remember: The Top of Data line will always have the known Line Pointer value of 1. You can use that fact when you are scanning the lines in your file, and you need a Line Pointer value as a starting point.

This means that you can scan a range of lines (special or ordinary) using a FOR/NEXT loop, checking the "type" of each line with an `Is_XXXX` function, incrementing the line pointer by 1

each time, and taking appropriate action as needed. The most common such test will be checking for **Is_Data**, passing the function a line pointer argument.

Like line numbers, valid line pointers will never be zero. However, whatever **other** values might be taken on by a line pointer, you should always let SPFLite inform you, and **not** make assumptions about the value you may get. Future releases of SPFLite could change those assumptions, and if you were relying on a particular implementation, your macro might not work right in a future release. Don't try to be too 'tricky' with line pointers; just use them as the documentation and examples show you.

What a line pointer definitely is **not** is a true memory pointer to some dynamically-allocated storage area within **thinBasic**. Don't even **think** of trying to use it that way.

The SPF_CMD function and Line Numbers vs. Line Pointers

The SPF_CMD function is used to issue SPFLite Edit Session primary commands. The command string you pass to the function looks just like one you would type in yourself on the primary command line. There are a few minor differences:

- SPF_CMD only takes one command. You cannot use a command separator character (like a ; semicolon) to perform multiple commands.
- The special ! notation that erases the primary command area prior to executing the command is not supported. That feature is primarily in place to assist key-mapped commands to function properly in case there were any "left over" commands or operands in the primary command area, an issue that doesn't apply to a macro.
- Presently, you cannot issue a macro call from a macro. So, in **AX.MACRO**, you cannot issue a call to **SPF_CMD("BX")** for instance.

When you create a command string for SPF_CMD using string constants and expressions, and the command contains a line reference, you must determine if the reference is a line **number** or a line **pointer**. Most macros will use line **pointers** almost exclusively.

If you have a Line Number value, you create the command string so that it appears the same as you would type it in on the primary command line. Recall that SPFLite allows what are called "temporary line labels" or "line number pseudo-labels", which are line numbers with a preceding dot.

For example, if you wanted to change ABCD to WXYZ on line 4 of the edit file, you could issue

```
SPF_CMD ("C ABCD WXYZ .4 ALL")
```

However, if you have a Line Pointer value, you can't (directly) use such notation. For example, the line with ABCD might have a line **number** of 4, but a line **pointer** of 7. You may have gotten this value of 7 from a function call like GET_FIND_LPTR. Now, if you created a command like this:

```
SPF_CMD ("C ABCD WXYZ ." + TSTR$(GET_FIND_LPTR) + " ALL")
```

it would evaluate to this:

```
SPF_CMD ("C ABCD WXYZ .7 ALL")
```

but **that won't work**, because it's referring to the wrong line. Since most SPFLite macro interface functions that deal with lines use line **pointers**, somehow we have to make SPF_CMD understand what we mean.

There's two ways to do this. First, you can use the conversion function GET_LNUM. In our example, that would convert the 7 into a 4:

```
SPF_CMD ("C ABCD WXZY ." + TSTR$(GET_LNUM (GET_FIND_LPTR)) +
" ALL")
```

Second, SPF_CMD allows an alternative notation. If you use an ! exclamation point **instead of a dot**, SPF_CMD will understand the value to be for a line **pointer** rather than a line **number**. When you do this, SPF_CMD will internally convert the line pointer for you into a line number.

In our example, you would do this:

```
SPF_CMD ("C ABCD WXZY !" + TSTR$(GET_FIND_LPTR) + " ALL")
```

and it would evaluate to this:

```
SPF_CMD ("C ABCD WXZY !7 ALL")
```

The SPF_CMD function would then see the exclamation point, take the number 7, internally call GET_LNUM itself to obtain the line number 4, and finally treat the command as if you issued:

```
SPF_CMD ("C ABCD WXZY .4 ALL")
```

Some points to keep in mind if you use the exclamation point notation:

- The exclamation point must be preceded by a blank, and the number after it must be followed by a blank or the end of the string
- Exclamation points are only recognized if not in SPFLite quotes.

So, a function call of `SPF_CMD ("C ABCD ' ! 7 ' ALL")` would have nothing to do with this notation; the `! 7` is just an ordinary string.

Line Pointers or Line Numbers: Which is Better?

Since there are two possible ways to refer to a line, which way is "better" ? To (probably) no one's surprise, the answer is, "it depends".

A more appropriate question is, better for **what**? It depends on what you're doing in your macro.

This question is similar to asking which is better, Celsius or Fahrenheit? It depends where you live. (Since George lives in Canada, and I live in the States, we each have our own preferences. Basically, whether it's 100 C or 212 F, you know you're in hot water ! - RH)

There are several factors to consider when making this decision:

- The SPFLite macro interface functions take line **pointers**. So, this is the form you will use most often, and if line pointers are what you have, this will be the most efficient method. For the most common cases, you will get a line number (or label) as a macro argument, convert it to a line pointer using Get_LPTR, and use that value from then on.

- The SPF_CMD function will accept either notation (as discussed above).
- If you have one form, but need another, you can call a conversion function to get the desired value.
- Line pointers may be somewhat transient, depending on what you're doing. For example, suppose you have several non-data lines in your file. These could be =PROF> lines, =NOTE> lines, =TABS> lines, etc. You may also have excluded-line placeholders present. If you obtain a line **pointer**, and then these special lines disappear as a result of issuing a RESET command or by direct deletion, the line pointer may no longer be valid. A line **number** will remain valid across a RESET command.
- It is important to understand that when a line **pointer** becomes "invalid" it does **not** mean you can't use it. For example, a line pointer might contain 5, meaning it's the fifth line from the beginning, counting the Top of Data as line-pointer line 1, but it would be the fourth data line. Now, if you inserted a =NOTE> or =TABS> line somewhere between the Top of Data and data line 4, or removed one already there, a line pointer of 5 won't point to data line number 4 any more. **But that's won't stop you from using it.** You can still pass a value of 5 to any of the macro interface functions that take line pointer arguments - the function won't stop you or claim the number 5 was "invalid". However, if you do that, the 5 wouldn't refer to data line 4 any more, but to some **other** line. That would almost certainly **not** be what you wanted.

The point is, line pointers and line numbers **both** have their place. You need to know what you're doing, and use them for what they are good for.

For line pointers, you must understand the rules that affect the validity of a line pointer reference, and stay within those rules. The main rule is that you must not "rock the boat" by doing anything that will change the relative line-pointer positions of data lines. Things that cause that are inserting and deleting (other) data lines, inserting or deleting special lines, and excluding or unexcluding lines. Special and excluded lines are affected by the **RESET** command. Provided you don't do anything to invalidate a pointer, you can continue to use it without problems. (They are not "fragile" and won't "break" by themselves.)

If you are forced to "rock the boat" so to speak, then you may have to recalculate new line pointers. Suppose you needed to do a **RESET** command that would impact the line pointer to a data line. You could "save" the pointer by (a) converting it to a line number, (b) issuing the **RESET** command, and finally (c) converting the line number back to a line pointer. You could also put a label on the line, and retrieve the label's line pointer later on.

As long as you keep these rules in mind and live within them, everything will work just fine.

Understanding Source and Destination Line Ranges

A macro will often need to refer to a line range. The way in which a macro obtains line-range information depends on whether it was launched as a line-command macro or as a primary-command macro.

Primary command macros may include line-range information as operands, such as line labels (including pseudo-label line numbers) and line tags. However, SPFLite will **not** interpret such operands. It merely passes them along to the macro, which must retrieve them with the Get_Arg\$ function and then interpret them as it sees fit. For line label operands, you would use the Get_LPTR function to convert them to a Line Pointer value, which is what most line-referencing macros use.

This means that if you create a macro called **ABC.MACRO**, you **can** issue a primary macro command like **ABC .AA .BB**, but **you** would have to write **thinBasic** code to interpret arguments 1 and 2 so that you could utilize the provided labels.

This may seem like an inconvenience (and to a certain extent, it is) but this design allows you complete control over how you would use these labels. For example, you might wish to write a macro that took 3 or even 4 labels. Standard SPFLite primary command processing would treat that as a syntax error, but your macro code could do something to address your unique requirements. If SPFLite recognized a standard line-range operand on macros (and nothing else), it would simplify some tasks, but would hold you back from doing other things. For now, the design seems like a good compromise.

While you don't have direct support for line labels on primary macro commands, you **do** have the ability to query other line commands that may be pending when the macro is launched. These are **source line ranges** and **destination line ranges**. Specifying source and destination line ranges is optional when launching a macro.

Technically, a line-command macro always defines its own source line range by its very presence in the line command area, so when you use a line-command macro, you always have an implicit source line range; it isn't optional.

For a primary-command macro, a **source** line range can be a **CC** or **MM** block. Such blocks can be single lines, a range of lines defined by a C or M followed by **n** or a **/** or **** modifier, or a pair of **CC** or **MM** commands to define a block.

For a line-command macro, the macro name itself appears on one or two data lines to define the **source** block. As discussed elsewhere, SPFLite automatically detects the single-line form vs. the block-mode form of a line command. For example, if you write a macro called AX, then you could do any of the following to define a source block: **AX**, **AXn**, **AX/**, **AX**, **AXX** or **AXAX**.

For both primary-command and line-command macros, you can specify a **destination line range** using any of the following line commands:

A, **AA**, B, **BB**, H, **HH**, W, **WW**, O, **OO**, OR and **ORR**.

The non-block forms **A**, **B**, **H**, **W**, **O** and **OR** can take the usual **n**, **/** and **** modifiers. You can also specify a trailing **+** or **-** modifier.

SPFLite does not act upon source or destination line command operands when used by a macro. That means that **CC** will **not** copy lines, **MM** will **not** move lines, **OO** will **not** overlay lines, **AA** will **not** intersperse copied data every few lines, **HH** will **not** replace lines, and so on.

But if that's true, what **does** it do? What **good** is it to use any of these things? These source and destination line ranges pass along information to the macro, which you can query and act on yourself. **You** can decide which of any of these line commands you want to recognize in your macro, and decide what they will do if used. What SPFLite does is (a) ensures that only these line commands can coexist with your macro's execution, and no other ones, (b) makes sure that they are in the proper format, and properly matched if used in pairs, and (c) will extract any addition information like **n** values and **+** or **-** and **&** modifiers that may be present, so you can look at them and act accordingly.

For the source line range, you have the following functions available:

<code>Get_Src_LCmd\$</code>	Get a string containing the name of the source line command. For line-command macros, this is the macro name
-----------------------------	--

	(possibly a plural/block form) of the macro name. For primary-command macros, this may be a C/CC or M/MM command, or may be an empty (null) string if no C/CC/M/MM was used with a primary-command macro.
<i>Get_Src1_Lptr</i>	Get the line pointer of the first (or only) line in the source line range, or 0 if not applicable.
<i>Get_Src2_Lptr</i>	Get the line pointer of the last (or only) line in the source line range, or 0 if not applicable.
<i>Get_Src_Op</i>	Get optional n value on the source line range. For example, if you have a source line range of C5 , Get_Src_Op will return the number 5 . The function returns 0 when no explicit command operand number is specified. For instance, if you define a source block such as CC which happens to be 5 lines long, the function will still return 0, because the number 5 was not actually specified on the CC block.
<i>Get_Src_LMOD\$</i>	<p>Gets the + or - (post-unexclude or post-exclude) modifier and & (Retain) modifier if one is present on the source line range. The returned value is always two characters long. Position 1 will contain + or - or blank, and position 2 will contain a R or blank.</p> <p>The / and \ line command modifiers are processed, if present, and this gets reflected in first and last LPTR values for the line range, but the / and \ codes themselves are not returned as LMOD values.</p>

For the destination line range, you have the following functions available:

<i>Get_Dest_LCmd\$</i>	Get a string containing the name of the destination line command. This may be an A/AA , B/BB , H/HH , W/WW , O/OO , or OR/ORR command, or may be an empty (null) string if no destination line range was specified.
<i>Get_Dest1_Lptr</i>	Get the line pointer of the first (or only) line in the destination line range, or 0 if not applicable.
<i>Get_Dest2_Lptr</i>	Get the line pointer of the last (or only) line in the destination line range, or 0 if not applicable.
<i>Get_Dest_Op</i>	Get optional n value on the destination line range. For example, if you have a destination line range of H5 , Get_Dest_Op will return the number 5 . The function returns 0 when no explicit command operand number is specified. For instance, if you define a destination block such as HH which happens to be 5 lines long, the function will still return 0, because the number 5 was not actually specified on the HH block.
<i>Get_Dest_LMOD\$</i>	Gets the + or - (post-unexclude or post-exclude) modifier

	<p>and & (Retain) modifier if one is present on the destination line range. The returned value is always two characters long. Position 1 will contain + or - or blank, and position 2 will contain a R or blank.</p> <p>The / and \ line command modifiers are processed, if present, and this gets reflected in first and last LPTR values for the line range, but the / and \ codes themselves are not returned as LMOD values.</p>
--	---

Note that when you use line macros in block form, or use destinations such as OR, you can define a 'block' by repeating the last character. (This is the same rule that ISPF uses, and we follow the ISPF rules on this.) So, the block form of a macro **AX** is **AXX**. When you ask for the line command name of the source or destination using Get_Src_LCmd\$ or Get_Dest_Lcmd\$, you get the actual command used, which in this case is **AXX**.

File Manager Interfacing

The macro interface to File Manager is considerably simpler than using the Edit Session interface. The screen content is simpler and the available SPFLite functions fewer and more straightforward.

The functions can be divided into three general groupings:

General File Manager Selection Criteria and Mode

The functions in this section are used to retrieve the current File Manager Mode and the criteria used to control the current display.

Functions are:

FMGet_Mode	Returns the general Mode of the File Manager display.
FMGet_FilePath\$	Returns the contents of the File Path field - EVEN IF the display is not in FilePath mode.
FMGet_FilePattern\$	Returns the current File Pattern mask - EVEN IF the display is not in FilePath mode.
FMGet_FListName\$	Returns the current FLIST name being displayed. If in FilePath mode, Null is returned.
FMGet_FCount	Returns the current number of line of file data being displayed.
FMGet_Folder_Class({pathname})	Returns the Backup / Normal class of the folder. If a Pathname provided, it returns the status of that folder. If no Pathname, and in FilePath mode, returns the status of the current FilePath.

Specific File Entry Details

The functions here are related to retrieving specific details about a specific line item entry from the File Manager display.

Functions are:

FMGet_Backup_Versions (FNum)	Returns the available # of Backup versions for the specified file at line number <i>FNum</i> .
FMGet_Cmd\$ (FNum)	Returns the current contents of the Line command area for the specified file at line number <i>FNum</i> .
FMGet_Extension\$ (FNum)	Returns the file extension (including the period) for the specified file at line number <i>FNum</i> .
FMGet_FileAttributes (FNum)	Returns the file attribute area for the specified file at line number <i>FNum</i> . This is a binary DWORD of bit flags describing the type of file system entry.
FMGet_File_Class (FNum)	Returns the general file class for the specified file at line number <i>FNum</i> . The Class is one of: a normal file; a Backup of a data file; or a Backup of an associated STATE file.
FMGet_FileName\$ (FNum)	Returns the filename for the specified file at line number <i>FNum</i> .
FMGet_FileSize (FNum)	Returns the size of the file specified at line number <i>FNum</i> . The value is returned as a QUAD integer
FMGet_FileSize\$ (FNum)	Returns the size of the file specified at line number <i>FNum</i> . The value is returned as a formatted string value. e.g. 9,999,999.
FMGet_FType (FNum)	Returns the type of File Manager entry for the line specified by <i>FNum</i> . e.g. Directory folder (Up or Down), file, FLIST entry etc.
FMGet_Lines (FNum)	Returns the number of Lines in the file (if available) for the file specified at line number <i>FNum</i> . If the Lines value is not available, -1 will be returned
FMGet_Note\$ (FNum)	Returns the contents of the Note field (if available) for the file specified at line number <i>FNum</i> . If the Note field is not available, a Null is returned.
FMGet_Path\$ (FNum)	Returns the Path to the file (including the trailing \) for the file specified at line number <i>FNum</i> .
FMGet_CreationTime\$ (FNum)	Returns the Creation time for the file specified at line number <i>FNum</i> . The format returned is: yyyy-mm-dd hh:mm
FMGet_LastWriteTime\$ (FNum)	Returns the Last Write time for the file specified at line

<code>me\$ (FNum)</code>	number <i>FNum</i> . The format returned is: yyyy-mm-dd hh:mm
--------------------------	---

Action Oriented Function Calls

These functions provide for requests for actions to be taken in the File Manager environment.

Functions are:

<code>SPF_FMLCmd (FNum, cmd-string)</code>	Execute a normal FM line command against a specified file at line number <i>FNum</i> . See the Function details for the supported line commands.
<code>FMSet_Cmd (FNum, string)</code>	Modify the line command field for a specified file at line number <i>FNum</i> .
<code>FMSet_Msg (FNum, string)</code>	Issue a message to temporarily overlay the filename for a file line specified by <i>FNum</i> .

Globally Stored Data

Introduction to globally-stored data

The SPFLite macro facility provides functions which allow you to save numeric and/or string data from your macro. This saved data can be retrieved later in the same macro, in a subsequent new execution of the same macro, or by any other macro executing later.

See [Function Details](#) for information on calling these functions.

Globally-stored data can enable the creation of a set of related macros and provide a means for them to communicate with each other. It could also be used to store "state-like" information for a single macro that is executed repeatedly over the course of an edit session.

The stored values are maintained **only** for the duration of the current SPFLite session. You may do multiple Edit/Save functions on different files, and the global data will be maintained. Globally-stored data will be discarded when SPFLite is terminated.

Globally-stored data is not "persistent" in the way that certain Profile and edit settings are when you have **STATE ON** in effect. Globally-stored data, and **STATE** information, are two completely different concepts.

If you really needed globally-stored data to persist across SPFLite executions, it is possible you could write out the data to a file and then read it back in later, using file-management services provided by thinBasic. We haven't experimented with this ourselves, but you're welcome to try. (If you do, let us know how it works out for you.)

Or you could use the SPFLite SET variable support to retain data using the [Get_SetVar\\$](#) and [Set_SetVar](#) functions.

Also be aware that globally-stored data is only "global" within a given SPFLite execution. If you have multiple instances of SPFLite running in Windows, each one has its **own** copy of globally-stored data for its own execution instance. The global data is **not** shared across multiple SPFLite instances.

The data is held in "storage pools" as a series of key/data pairs. (Some people refer to that as a "map" or "associative storage", if that helps.) The keys may be any strings of your choosing. There are two global storage pools. One is for storing string data, and the other is for storing numeric data.

Like global variables in any other language, globally-stored values can be a good thing if handled carefully, but could cause problems if misused in a poorly designed macro, for reasons well-known to experienced software developers. Persistent data sometimes causes unexpected side effects, which you need to be aware of, and plan for. Generally, you should only use globally-stored data if conventional local variables are not enough. They should be used sparingly, and only as a last resort. (As they say, be careful what you wish for.)

Sub-Tables

Since there are just two Global pools (String and Numeric), you may have multiple requirements for storage of a single type. This is handled by using an Optional sub-table number to create unique groupings. You may use any number you like to identify a table, and the same key-names may occur in different subtables. The Table-number is entered as an

optional first parameter to the Set_Gbl, Get_Gbl and Reset_Gbl functions. If the operand is omitted, a value of zero is assumed.

Examples:

<code>SET_GBL_NUM("KEY1", 12)</code>	will store 12 as a value for the key KEY1 in the default Table zero
<code>SET_GBL_NUM(4, "KEY1", 15)</code>	will store 15 as a value for the key KEY1 in Table 4
<code>GET_GBL_NUM("KEY1")</code>	will return 12 as a value for the key KEY1 from the default Table zero
<code>GET_GBL_NUM(2, "KEY1")</code>	will return 0 as a value since it asked for a value from Table 2 which was never used.
<code>GET_GBL_NUM(4, "KEY1")</code>	will return 15 as a value for the key KEY1 from Table 4

Undefined keys and deletion of global storage

If you request the value of an undefined key, the functions will return default values of **0** (for Get_Gbl_Num) and a null string (for Get_Gbl_Str\$). Thus, it is not necessary to "initialize" global storage if these default values are sufficient.

You can also use the fact that an undefined global number returns a zero value as a "flag" showing that some (other) global value needed to be initialized. Recalling that FALSE is the same thing as zero, here is an example of how to do that:

```
IF GET_GBL_NUM("INITIALIZED") = FALSE THEN
    SET_GBL_STR ("MY-STR-VAR", "some string value")
    SET_GBL_NUM ("MY-NUM-VAR", some-numeric-value)
    SET_GBL_NUM ("INITIALIZED", TRUE)
END IF
```

At present, there is no way to delete an individual key/value pair from a storage pool, once it's defined. To achieve the effect of a deletion, you can store a zero or null value. If it's necessary to determine whether a zero or null is an actually-stored value or just the default value returned when querying for an undefined key, you can check the return code. Querying for a defined key will produce a return code of **0**, and querying for an undefined key will produce a return code of **8**. You can delete **every** global number or string using the functions using [Reset_Gbl_Num](#) and [Reset_Gbl_Str](#). These two functions also enable you to delete only a specific sub-table. See the Function details for further assistance.

Example

If you wanted to store the last modified date for the current edit file, you could do so by issuing:

```
Set_Gbl_Str(Get_FileName$, Get_FileDate$)
```

This would create a global variable with a key equal to the filename, and a value equal to the file's modification date. At some future time this macro, or some other macro, could retrieve this particular date for the file by issuing:

```
Date-var = Get_Gbl_Str$("the file name")
```

If still had the same data file actively being edited, you could do this:

```
Date-var = Get_Gbl_Str$(Get_FileName$)
```

There is no limit to the number of global variables, or the size of the saved strings, other than available system memory.

Available global functions

See the relevant section in [Function Details](#) for additional information on these functions.

<code>Set_Gbl_Num([TblNum,] key-str, value-num)</code>	Stores the value-num under the key specified by key-str . If a key matching the contents of key-str already exists, the numeric value it was previously associated with is replaced with the new value-num . If a key matching the contents of the key-str does not already exist, it is created.
<code>Set_Gbl_Str([TblNum,] key-str, value-str)</code>	Stores the value-str under the key specified by key-str . If a key matching the contents of key-str already exists, the string value it was previously associated with is replaced with the new value-str . If a key matching the contents of the key-str does not already exist, it is created.
<code>num-var = Get_Gbl_Num([TblNum,] key-str)</code>	Returns the numeric value associated with key-str . If the key does not exist, zero is returned, and an error is stored in the RC and Msg\$ variables.
<code>str-var = Get_Gbl_Str\$([TblNum,] key-str)</code>	Returns the string value associated with key-str . If the key does not exist, a null (empty) string is returned, and an error is stored in the RC and Msg\$ variables.
<code>num-var = Get_Gbl_Str_Count</code>	Returns the current number of variables stored in the String pool.
<code>num-var = Get_Gbl_Num_Count</code>	Returns the current number of variables stored in the Numeric pool.
<code>str-var =</code>	Returns the key name of the specified

<pre>Get_Gbl_Str_Name\$(index- num)</pre>	<p>item in the String pool. index-num must be a number in the range of 1 through the the value of Get_Gbl_Str_Count.</p> <p>The keys stored in the String pool are not stored in any predicable order. To find the location of a given key in the pool, you must do a linear search from beginning to end until it is found. Pool management is done using software outside the control of SPFLite. Adding new entries to a pool may change the index of previously stored entries. If this happens, you may need to perform a new search to redetermine the location of a previously-located key.</p>
<pre>str-var = Get_Gbl_Str_TableName\$(index-num)</pre>	<p>Returns the Table-number and key name of the specified item in the String pool. index-num must be a number in the range of 1 through the the value of Get_Gbl_Str_Count.</p> <p>The str-var returned contains the table-number preceding the key name, separated by a comma. e.g. a value like 4,KEYWORD.</p> <p>The table number is easily accessed via VAL(str-var) The keyname can be accessed via REMAIN\$(str-var, ",")</p> <p>See also the note above re: order of entries.</p>
<pre>str-var = Get_Gbl_Num_Name\$(index- num)</pre>	<p>Returns the key name of the specified item in the Numeric pool. index-num must be a number in the range of 1 through the the value of Get_Gbl_Num_Count.</p> <p>The keys stored in the Numeric pool are not stored in any predicable order. To find the location of a given key in the pool, you must do a linear search from beginning to end until it is found. Pool management is done using software outside the control of SPFLite. Adding new entries to a pool may change the index of previously stored entries. If this happens, you</p>

	may need to perform a new search to redetermine the location of a previously-located key.
<pre>str-var = Get_Gbl_Num_TableName\$ (<i>index-num</i>)</pre>	<p>Returns the Table-number and key name of the specified item in the String pool.</p> <p><i>index-num</i> must be a number in the range of 1 through the the value of Get_Gbl_Str_Count.</p> <p>The str-var returned contains the table-number preceding the key name, separated by a comma. e.g. a value like 4,KEYWORD.</p> <p>The table number is easily accessed via VAL(str-var)</p> <p>The keyname can be accessed via REMAIN\$(str-var, ",")</p> <p>See also the note above re: order of entries.</p>
<pre>num-var = Reset Gbl Num</pre>	<p>Clears the entire Gbl_Num storage area. Returns True is successful, or False if the Clear fails.</p>
<pre>num-var = Reset Gbl Str</pre>	<p>Clears the entire Gbl_Str storage area. Returns True is successful, or False if the Clear fails.</p>

Working with Line Colors

Overview of SPFLite color support in Edit Sessions

SPFLite provides primary commands and keyboard primitive functions to support what are called Virtual Highlighting Pens.

A highlighting pen can color a portion of text in one of fifteen colors:

The available keyboard primitive functions are:

(Pen/colorname)
(Pen/Std)

You can specify a color on **FIND** and **CHANGE** commands. You can also use **LOCATE** to find lines having a particular color, and can use **RESET** to clear the color from all lines.

See the main SPFLite Help documentation for more details on these commands and primitive functions.

Macro functions for color support

The color interface for macros provides a means to query the current color state of a line, and a way to alter that state.

What could you do with this capability? You may want to use it to analyze a block of text, and add or remove highlighting colors based on some particular condition or data value you are interested in. You could look for, or change, the text coloring in ways that are beyond the ability of native SPFLite FIND and CHANGE commands.

Recall that text highlighting pens come in up to 15 colors: The current color of a character position in a given line is represented by a unique single character. These letters and their color name are:

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

Character positions having the standard text color are presented by a blank.

Suppose you have a line of data colored like this. (The colors are shown using one possible

color palette selection. You can use the Screen tab of the Global Options dialog to choose any desired text foreground and background for each of the possible colors.)

000001 LINE ONE OF COLOR-HIGHLIGHTED DATA

To "get" the color of this line, you ask for a string that represents the color of each column of data on that line. Assume that **currLptr** is the line pointer to that line, and **Clr_Line** is a string variable:

```
Clr_Line = Get_Clr_Line$ (currLptr)
```

This would produce an "array" of color codes, one character for each character in the line of data, as a string value. Here's what the function would return:

```
data line = "LINE ONE OF COLOR-HIGHLIGHTED DATA"
color line = "      RRR                      BBBB"
```

Because the data line has a length of 34, the string returned in the color-line variable is also of length 34.

Now, suppose you wanted to set the text value "HIGHLIGHTED" to Yellow, and make the word "ONE" the standard color. You would pass a "new" color array to the function **Set_Clr_Line**. This function **completely replaces** all existing colors for that line. So, the typical way you would do this is:

- Call **Get_Clr_Line\$** to find the current colors for the line
- Use **thinBasic** statements to modify the color-string as needed.
- In all likelihood, color changes will involve the use of the **MID\$** function and/or the **MID\$** statement. See the **thinBasic** documentation for more details on these features.
- Call **Set_Clr_Line** to replace the colors.
- Be aware that the word COLOR is a reserved word in **thinBasic**; you can't use it as a variable name.

Assume that you have the line of data shown above, and the variable **currLptr** contains a Line Pointer value to that line. To remove the red color from the word "ONE", and make "HIGHLIGHTED" display as Yellow, you'd write statements like the following:

```
DIM Clr_Line AS STRING
Clr_Line = Get_Clr_Line$ (currLptr)
MID$(Clr_Line, 6, 3) = "      " ' remove color
from ONE
MID$(Clr_Line, 19, 11) = "YYYYYYYYYYY" ' make HIGHLIGHTED
display as Yellow
Set_Clr_Line (currLptr, Clr_Line)
```

where

```
data line = "LINE ONE OF COLOR HIGHLIGHTED DATA"
color line = "      YYYYYYYYYY BBBB"
```

Note that "DATA" was Blue before, and we are setting to Blue "again". That's OK. We also changed "ONE" from Red to "standard" by making its color-code positions blank, and we changed the word "HIGHLIGHTED" from standard to Yellow. Observe that each time you call **Set_Clr_Line**, you are performing a complete replacement of the colors of every character on the data line.

Rules:

- Only data lines have a color interface, not special lines (like TABS, BNDS and NOTE lines). If the LPTR is not for a valid, existing data line, RC is set to 8, and no other action is taken. You cannot change the color of special lines.
- **Get_Clr_Line\$** always returns a string of length identical to the data line, even if that length is zero.
- You are not allowed to attempt to change the color of non-existent character positions of a data line. The string passed to **Set_Clr_Line** cannot be longer than the data line you are trying to update, which is another way of saying that it cannot be longer than the string returned by [Get_Clr_Line\\$](#) for that same data line. If [Set_Clr_Line](#) does get a string longer than the data line, RC is set to 8 and the colors on the data line are not changed. For this reason, it is best to always fetch the current color state of a line into a string using **Get_Clr_Line\$**, and then modify that string, being careful not to extend its length. The function [Get_Line_Len](#) may be useful in this regard.
- You should normally complete any macro modifications to the data line before proceeding with changes to the associated color line. This will ensure the line length problem will not occur.
- If the color string passed to [Set_Clr_Line](#) is shorter than the data line, the color string is effectively treated as if padded with blanks to make it as long as the data line.
- If any color codes other than the valid ones (or blank) are passed to [Set_Clr_Line](#), RC is set to 8, and no other action is taken.
- It is not necessary for **STATE ON** to be in effect if **Set_Clr_Line** is used. However, unless **STATE** is **ON**, any changes to the text colors will not persist after the edit file is closed.
- For [Get_Clr_Line\\$](#), the color code letters are returned in upper case. For [Set_Clr_Line](#), you may specify these codes in either upper or lower case.

Working with the Clipboard

thinBasic allows you to access the Windows clipboard. You can use this fact to gain access to data placed into the clipboard during editing when a macro is executing.

To get the contents of the clipboard:

```
DIM clip AS STRING
clip = Clipboard_GetText
```

To change the contents of the clipboard:

```
Clipboard_SetText (stringExpression)
```

Keep in mind that when you copy text into the clipboard, it will normally be stored with a trailing CR/LF pair. SPFLite handles this automatically when you are editing, but if you directly access the clipboard in a **thinBasic** macro, you are going to see a CR/LF pair for every line in the clipboard - even when you only copied a single word from a single line.

If this an issue, there are few things you can do to deal with this:

- You can KEYMAP a key or mouse button to (CopyPasteRaw), which copies data to the clipboard without the CR/LF line terminator.
- You can scan the returned string, and trim off the CR/LF

Example:

```
DIM clip AS STRING
clip = Clipboard_GetText
if Right$(clip,2) = $CRLF then
    clip = Remove$(clip, $CRLF)
end if
```

This example would remove ALL instances of CR/LF, even ones embedded in the middle if the clipboard had more than one line.

If you are going to handle multi-line clipboards, you will probably need more involved macro coding than this, but for simple strings this should be enough to get by.

Debugging your Macro

Contents of Article

[Tracking your macro's flow of control](#)

[Functions and Return Codes](#)

[SPFLite Function Trace](#)

[thinBasic Output Statements](#)

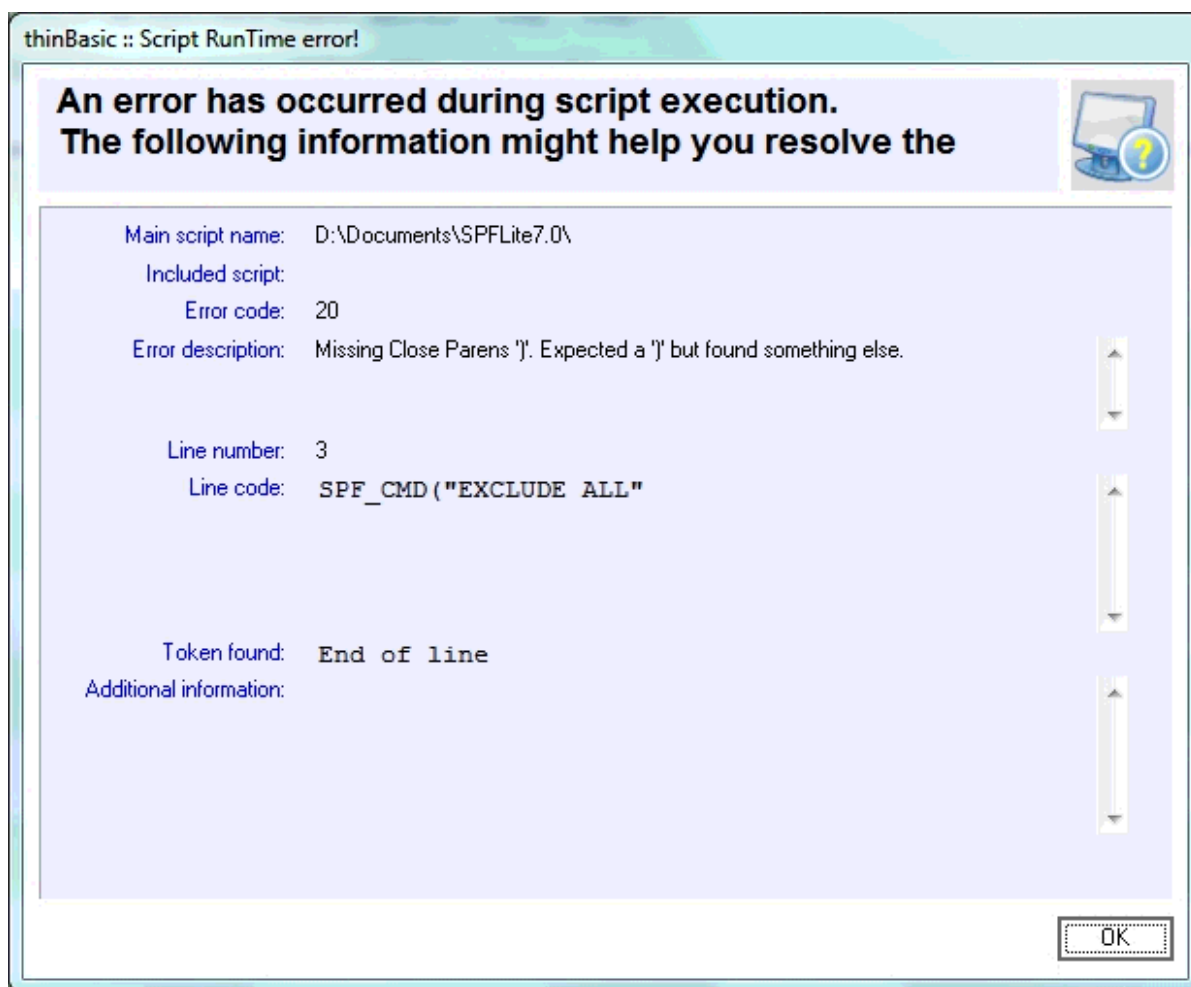
[The thinBasic Trace Module](#)

Introduction

Just as with any programming exercise, your macro may not do just what you want on the first attempt.

So, what types of errors are you likely to see? **Simple syntax errors** are usually the simplest to correct.

thinBasic provides you with a detailed error display like this:



This message tells you what **thinBasic** expected to find - a closing parenthesis - and what it found instead - an End of line.

Once you add a closing) to line 3 this problem is fixed.

However, there are some errors that thinBasic detects that result in misleading and confusing messages:

- Duplicating reserved words as variable names can trigger strange errors. If the error is reported immediately following a DIM statement that declared variables, review the variable names you have used.
- Delimiter parsing errors (expecting a closing parenthesis and not finding it) will sometimes describe the 'found' tag as some value from the previous valid statement.
- Other missing syntax, like a missing close-quote, will cause similar confusing messages

These can be frustrating, but the line number flagged as in error is usually where the problem line is, or is close to it.

The other error most commonly made is misspelling of function names, and a quick browse of the Help file should provide the correct spelling. An easy mistake to make is failing to put a \$ for a string-based function like **Mid\$**. If you do that, you'll get an error message about code that "seems" to be correct but isn't. Similar to misspelled names is failing to define a variable before you used it.

Tracking your macro's flow of control

A common debugging problem is code that does not follow the path you think it should. The macro may just end prematurely, or seems to be in an area of code that it should not be, and you have to figure out: How did the macro end up **there?**

The function `SPF_Debug` is useful in a variety of ways in solving such problems. It allows you to write messages to a debug log . These messages can be anything from simple 'breadcrumb' messages, like:

```
SPF_Debug("Starting line loop")
SPF_Debug("Parameters validated")
```

to displays of macro variables, which can be done like this:

```
SPF_Debug("Operand1=" + Operand1)
```

Note: See the [CLS](#) command for information on how to clear or close the Debug Window

Functions and Return Codes

All of the SPFLite support functions will (in addition to the defined values returned by the functions) set a Return Code value and a Return Message value. You can get these values using the functions [Get_RC](#) and [Get_Msg\\$](#).

Good programming practice suggests that after calling any function that indicates success or failure with a return code, the return code should be tested and acted on. In a perfect world that's a good idea, but in practice we usually only check a few of the results of function calls when it seems important. And, many of the SPFLite interface calls are simple information retrieval calls where the functions never (or hardly ever) fail, and checking a return code would

be unnecessary.

In addition, using nested function calls, like `Get_LPtr (Get_Arg$(1))` would be very burdensome if we had to break down each function call into single steps and add RC tests. But, when something is not working, and you suspect a failing function call, what option do we have, other than breaking things down into single-step code?

SPFLite Function Trace

The answer is to activate function tracing using `SPF_Trace`, which is called as follows:

```
SPF_Trace( [ ON | OFF | ERROR ] )
```

If you insert an `SPF_Trace(ON)` call at the beginning of your macro code (after the macro prototype line) the macro processor will create a debug log window, and write a trace line for every SPFLite interface function call. It will show the function call and its calling parameters, along with the resultant Return code, Error message text and the returned value (if appropriate).

Lets look at an example. Given the following macro:

```
' Try.MACRO aaa bbb ccc ddd eee fff ggg hhh
SPF_Debug("Args: " + Get_Arg$(4, 2))
halt
```

The macro writer tried to display some of the calling arguments using an `SPF_Debug` display function. But when it was run, all that displays is "Args:" and nothing else. Obviously, it seems like `Get_Arg$` is not working, but why? And how can we be be sure?

We could track this down the down the "hard way" with a brute-force approach, and break the macro down to individual statements, adding code to fetch and display error messages and return codes at every point - but that would be a lot of work. Instead, let's try using `SPF_Trace` and see how it can help. We add the `SPF_Trace` activation call, and the macro now looks like this:

```
' Try.MACRO aaa bbb ccc ddd eee fff ggg hhh
SPF_Trace(ON)
SPF_Debug("Args: " + Get_Arg$(4, 2))
halt
```

When run, the trace output looks like:

```
Get_Arg$(4, 2) RC=8 Invalid ARG indexes
Args:
SPF_Debug(Args: ) RC=0
```

There in the first line is our problem. The `Get_Arg$` is failing. Now we need to determine why. (By the way, the actual log won't highlight the display in yellow like this. We just added the color in the documentation here to make it stand out.) Reviewing the documentation for the [Get_Arg\\$](#) function reveals the problem. The two arguments are reversed; we needed to say (2, 4) and not the other way around.

Moral of the story: You have to read the manual.

What are the other two lines?

The `Args :` line is the output from the `SPF_Debug ()` function, `Debug` and `Trace` share the same debugging window.

The last line is the trace output for the **execution** of the `SPF_Debug ()` showing that it worked correctly (`RC=0`).

The `Trace` function can be turned ON and OFF selectively throughout your macro, which is handy when you can isolate the problem area to one section. If you don't need to trace an entire macro's execution, it will make the debugging output smaller, which makes it easier to follow.

The `ERROR` option requests that trace records need only be displayed when a function's return code is non-zero. This can reduce the volume of output down considerably, by turning the debugging output into "exception reporting", since it's usually more important to know when functions failed than when they succeeded.

Between `SPF_Trace` and `SPF_Debug`, it usually becomes simple to nail down the cause of most bugs.

Note: See the `CLS` command in the main SPFLite Help document for information on how to clear or close the Debug Window

thinBasic Output Statements

If you wish, you can use **thinBasic**'s output facilities to write debugging logs or any other information you wish. To do this, you must have the `CONSOLEthinBasic` module available at runtime.

You should place a statement of the form,

```
USES "CONSOLE"
```

prior to writing to the **thinBasic** console screen.

The "print" statements are slightly different than other versions of Basic you may know. They do not use semi-colons to influence the formatting of the output.

The **PRINT** statement will output one or more (comma-separated) items, but will **not** write an end-of-line termination (no Carriage Return and Line Feed).

The **PRINTL** statement ("print line") will output one or more (comma-separated) items, and **will** write an end-of-line termination.

You may also issue a **CLS** statement to clear the screen.

There are additional Console capabilities which you may find useful. See the **thinBasic** documentation under **thinBasic Modules - Console** for more information.

The thinBasic Trace Module

If you read the **thinBasic** documentation, you will notice in the Modules section they mention a Trace module, which for stand-alone **thinBasic** use provides a single-step debugging mode.

Unfortunately, the Trace module cannot be used presently in SPFLite Macro mode. If you

attempt to use this, crashes will occur. When this problem is resolved and **thinBasic** Trace functions correctly, we will announce its availability.

Function Overview

This section provides an overview of the various interface functions and their usage. For full details of syntax, allowed parameters, return values etc. click on the Function Name on the left to switch to the detailed description for the function.

Understanding SPFLite status values

A macro you run from your edit session, by placing a macro name on the primary command line, or on the sequence area of a data line, will return a **status** when it completes. A **status** consists of two values: a numeric **Return Code** value (often referred to as **RC**), and a **Message** string. This status is displayed on the upper-right portion of the edit screen, in the form of the Message string, and an optional Warning or Error message prefix in case a non-zero Return Code has been set.

When you call an SPFLite-supplied interface function (such as **Get_Line\$**), that function will **also** return a status. You may retrieve this status using the functions **Get_RC** and **Get_Msg\$**.

There are thus **two** sets of status information. The status returned to **you** (as an edit user that executed a macro command) that is visible on the edit screen as a message is called the **top-level status**. The status returned to a user-defined macro by a call to an SPFLite interface function is called the **function-level status**.

It is important to understand that the top-level status and the function-level status are **completely different and separate values**.

This means that if you issue a call to **Set_Msg(my_RC, my_Message)** it will change the top-level status values for the RC and Message, but will **not** change the function-level values that are returned by **Get_RC** and **Get_Msg\$**. Another way to look at this is that the function-level status is "lower" than the top-level status, and that status values can go "up" (via **Set_Msg**) but cannot go "down".

Example:

```
DIM rc as NUMBER
DIM msg as STRING
' ... some macro code
Set_Msg (8, "user defined error occurred")
rc = Get_RC
msg = Get_Msg$
' at this point, rc = 0, and msg = "", not the values set
by Set_Msg
```

The other key point to keep in mind is that SPFLite interface functions do not "communicate" the function-level RC and Message values between themselves. So, if one SPFLite function set an RC value of 8, the next SPFLite function you call will know nothing about that; it won't "look" at the RC=8 value set by the prior function call. Each function "starts over from scratch" in terms of setting the function-level status values, by assuming each time that the RC will be 0 and the Message will be an empty (null) string, unless conditions require them to be changed.

Return codes and returned messages from SPF-supplied functions

All SPF-supplied functions set the function-level Return Code (RC) variable and the standard Message text appropriately based on the success of the function. **This is done in addition to providing the return value from the function directly.** In a few cases, the value returned by the function will be the RC value itself, but most functions will return values other than the RC.

Functions built into **thinBasic** do **not** set the RC and message values. That only applies to functions we provide as part of the SPFLite macro interface. So, if you call a **thinBasic** function like **MID\$** to get a substring of a value, that does **not** change the RC or message.

The function-level status values can be obtained with the **Get_RC** and **Get_Msg\$** functions. A successful return will usually result in RC=0 and a message text of "" (null). However, some successful functions may result in a RC=0 but with a completed message text string. For example, invoking an **SPF_Cmd ("DCB ?")** function may result in a RC=0, but a **Get_Msg\$** return string of "DCB set to RECFM=U, EOL=CRLF, LRECL=0".

When a function returns a value which is not important to you, you can call it directly without assigning its returned value to a variable. For example, you can call

```
RC = SPF_Exec("MyProgram")
```

but if the return code isn't important, you can shorten this to:

```
SPF_Exec("MyProgram")
```

Return codes and returned messages from Set_Msg to the edit user

The function **Set_Msg** is used to pass back a top-level return code and message to the edit user (you) that is displayed in the upper-right corner of the edit screen.

It is important to understand that function-level RC and message values passed back by other functions (like **Get_Line\$**, for instance) are **not** automatically passed through to the edit user, even when the RC value is non-zero, signifying a warning or error condition was detected. You must explicitly call **Set_Msg** yourself if you wish a top-level status message to be displayed back to the user once the macro completes execution.

The values set by **Set_Msg** will be overridden by any other subsequent call to **Set_Msg**. If you call this function more than once, only the RC and Message of the last call will be used as the top-level status, and displayed on the edit screen when the macro completes.

If you execute a macro that never calls **Set_Msg**, it is treated the same as if **Set_Msg(0, "")** were called; that is, no message will appear on the edit screen when the macro completes.

Supposing you wanted to "pass along" the function-level status values up to the top-level status after an SPFLite function call, it would require you to call the **Set_Msg** function. If you did this, the call would take the form **Set_Msg(Get_RC, Get_Msg\$)**.

Example:

```
SPF_Cmd ("LINE 'R5' .1234")
```

```

Set_Msg (Get_RC, Get_Msg$)
' at this point, the visible top-level RC and Message
are
' set to the status returned by the SPF_Cmd function call

```

List of available macro functions

With the introduction of macro support for File Manager, it meant a new series of functions were created to interface with File Manager. Obviously, if these functions are invoked by a macro running in a normal Edit session, they would not work. Similarly, many of the original functions designed for an Edit session will not function in the File Manager tab.

So we have three classes of functions:

- Those that will work in all environments
- Those that will work only in File Manager
- Those that will work only in Edit sessions.

SPFLite will check for these conditions and terminate a macro which is using a function in the incorrect environment.

The functions here are divided by categories of interest, for ease of lookup.

	Functions for handling command line arguments	FM	Edit
<i>Get_Arg\$(n)</i> <i>Get_Arg\$(f,n)</i>	Returns a specified macro operand number n , or a range of operand numbers separated by single blanks.	Yes	Yes
<i>Get_Arg_Count</i>	Returns the number of supplied macro arguments.	Yes	Yes
<i>Get_MacName\$</i>	Returns the name of the current macro	Yes	Yes
<i>Get_Primary_Cmd\$</i>	Returns the name of the last issued Primary command.	Yes	Yes
<i>SPF_Parse</i>	Performs full operand parsing of command operands	Yes	Yes
<i>Get_Arg_KW</i>	Returns True/False based on a Keywords presence in the macro command line	Yes	Yes
<i>Get_Arg_KWGroup\$</i>	Returns the entered keyword from a list of mutually exclusive items	Yes	Yes
<i>Get_Arg_LRef\$(n)</i>	Returns the requested relative Line Reference from the macro command line	Yes	Yes
<i>Get_Arg_LRef_Count</i>	Returns the number of Line Reference type macro operands.	Yes	Yes
<i>Get_Arg_Tag\$(n)</i>	Returns the requested relative Tag Operand from the macro command line	Yes	Yes

<code>Get_Arg_Tag_Count</code>	Returns the number of Tag type macro operands.	Yes	Yes
<code>Get_Arg_TextLit\$(n)</code>	Returns the requested relative Text Literal from the macro command line	Yes	Yes
<code>Get_Arg_TextLit_Count</code>	Returns the number of Text Literal type macro operands.	Yes	Yes
<code>Get_Arg_NumLit\$(n)</code>	Returns the requested relative Numeric Literal from the macro command line	Yes	Yes
<code>Get_Arg_NumLit_Count</code>	Returns the number of Numeric Literal type macro operands.	Yes	Yes
<code>SPF_Quote\$</code>	Returns a string value properly enclosed in quotes	Yes	Yes
<code>SPF_UnQuote\$</code>	Returns a string value with any existing paired outer quotes removed.	Yes	Yes
<code>SPF_InvChar\$</code>	Returns a string with invalid display characters 'cleaned up'	Yes	Yes
<code>Get_FullPath\$(file-name)</code>	Returns a fully qualified path for a filename.	Yes	Yes
<code>Is_Line_Cmd</code>	Returns TRUE if macro launched as a line-command macro, otherwise FALSE	Yes	Yes
<code>Is_Primary_Cmd</code>	Returns TRUE if macro launched as a primary-command macro, otherwise FALSE	Yes	Yes
<code>Is_FM</code>	Returns TRUE if macro launched in the File Manager tab, otherwise FALSE.	Yes	Yes
<code>Set_Msg([rc-num,] msg-str)</code>	Sets the visible (top-level) return code and message that is returned to the user	Yes	Yes
<code>Halt([rc-num,] msg-str)</code>	Sets the visible (top-level) return code and message that is returned to the user and then immediately terminates the macro.	Yes	Yes
<code>Get_RC</code>	Contains the Return Code from the last executed SPFLite function call.	Yes	Yes
<code>Get_Msg\$</code>	Contains the Return Code from the last executed SPFLite function call.	Yes	Yes
<code>Get_EXE_Path\$</code>	Returns the full path of the folder where SPFLite.EXE resides.	Yes	Yes

<code>Get_INI_Path\$</code>	Returns the full path of the SPFLite data storage folder.	Yes	Yes
<code>SPF_Shell(cmd-str)</code>	Execute the provided command under the System's CMD.EXE command processor.	Yes	Yes
<code>SPF_Exec(cmd-str)</code>	Execute the provided command directly, not under the CMD.EXE command shell.	Yes	Yes
<code>SPF_Post_Do(Do-string)</code>	Specify a standard DO macro string which will be executed immediately when this macro has terminated.	Yes	Yes
	Functions that manage globally-stored values	FM	Edit
<code>Delete_Gbl_Num([Tblnum,]key-str)</code>	Deletes a current item (key-str) from the specified table (Tblnum)	Yes	Yes
<code>Delete_Gbl_Str([Tblnum,]key-str)</code>	Deletes a current item (key-str) from the specified table (Tblnum)	Yes	Yes
<code>Set_Gbl_Num([TblNum,]key-str,value-num)</code>	Stores the numeric <i>value</i> under the associated string <i>key</i> .	Yes	Yes
<code>Set_Gbl_Str([TblNum,]key-str,value-str)</code>	Stores the string <i>value</i> under the associated string <i>key</i> .	Yes	Yes
<code>Get_Gbl_Num([TblNum,]key-str)</code>	Retrieves the numeric value associated with <i>key</i> .	Yes	Yes
<code>Get_Gbl_Str\$([TblNum,]key-str)</code>	Retrieves the string value associated with <i>key</i> .	Yes	Yes
<code>Get_Gbl_Num_Count([TblNum])</code>	Returns the current number of variables stored in the Numeric pool.	Yes	Yes
<code>Get_Gbl_Str_Count([TblNum])</code>	Returns the current number of variables stored in the String pool.	Yes	Yes
<code>Get_Gbl_Num_Name\$(index-num)</code>	Returns the key name of the specified item in the Numeric pool.	Yes	Yes
<code>Get_Gbl_Num_TableName\$(index-num)</code>	Returns the Table-number and key name of the specified item in the Numeric pool.	Yes	Yes
<code>Get_Gbl_Str_Name\$(index-num)</code>	Returns the key name of the specified item in the String pool.	Yes	Yes
<code>Get_Gbl_Str_TableName\$(index-num)</code>	Returns the Table-number and key name of the specified item in the String pool.	Yes	Yes
<code>Reset_Gbl_Num([T</code>	Deletes the entire contents of the Gbl_Num	Yes	Yes

<code>blNum])</code>	storage area; all Keys and Values are deleted		
<code>Reset_Gbl_Str([T blNum])</code>	Deletes the entire contents of the Gbl_Str storage area; all Keys and Values are deleted	Yes	Yes
	Functions that manage environment and SET variables	FM	Edit
<code>Get_EnvVar\$(env-var-str)</code>	Returns the specified Windows Environment Variable data.	Yes	Yes
<code>Get_Instance\$</code>	Returns the name of the Instance for the current SPFLite session.	Yes	Yes
<code>Get_SETVar\$(set-var-str)</code>	Returns the specified variable value from the SPFLite SET variable pool.	Yes	Yes
<code>Set_SETVar(set-var-str, value-str)</code>	Sets the specified variable value in the SPFLite SET variable pool.	Yes	Yes
	Functions that manage macro debugging	FM	Edit
<code>SPF_Debug(msg-str)</code>	Display a message on the debugging console	Yes	Yes
<code>SPF_Loop_Check(ON OFF)</code>	Fetch and Set the SPFLite Macro loop monitoring function ON or OFF.	Yes	Yes
<code>SPF_Trace(mode-num)</code>	Set SPFLite trace mode to ON, OFF or ERROR	Yes	Yes
	Functions that manage the Edit file	FM	Edit
<code>Get_FilePath\$</code>	Returns the current file path for the current file.	No	Yes
<code>Get_FileBase\$</code>	Returns the base filename of the current edit file.	No	Yes
<code>Get_FileName\$</code>	Returns the filename of the current edit file.	No	Yes
<code>Get_FileExt\$</code>	Returns the current file extension.	No	Yes
<code>Get_FileDate\$</code>	Returns the last modified date of the current file.	No	Yes
<code>Get_FileTime\$</code>	Returns the last modified time of the current file.	No	Yes
<code>Get_First_LPtr</code>	Returns a Line Pointer to the first line of the Edit session	No	Yes

<i>Get_Last_LPtr</i>	Returns a Line Pointer to the last line of the Edit session	No	Yes
<i>Get_Select_First_LPtr</i>	Returns a Line Pointer to the first line of a selected text block	No	Yes
<i>Get_Select_Last_LPtr</i>	Returns a Line Pointer to the last line of a selected text block	No	Yes
<i>Get_Select_Col</i>	Returns a number indicating the leftmost selected column	No	Yes
<i>Get_Select_Len</i>	Returns a number indicating the length of the selected data	No	Yes
<i>Get_EOL_STR\$</i>	Returns a string containing the EOL specified in the current Profile	No	Yes
<i>Get_Profile\$</i>	Return a PROFILE setting as a string.	No	Yes
<i>Get_MARKLine\$</i>	Returns NONE or the current MARK line contents	No	Yes
<i>Get_BNDSLine\$</i>	Returns the current BNDS line.	No	Yes
<i>Set_MARK</i>	Sets a new value for the MARK line	No	Yes
<i>Get_MASKLine\$</i>	Return the current MASK line contents	No	Yes
<i>Set_MASK</i>	Sets a new value for the MASK line	No	Yes
<i>Get_TABSLine\$</i>	Return the current TABS line contents	No	Yes
<i>Set_TABS</i>	Sets a new value for the TABS line	No	Yes
<i>Get_WordChar\$</i>	Return the current WORD line contents	No	Yes
<i>Set_WORD</i>	Sets a new value for the WORD line	No	Yes
	Functions that manage the Edit session	FM	Edit
<i>Add_Array(line- ptr, array-name)</i>	Adds all lines in an array to the edit session.	No	Yes
<i>Add_Line(line- ptr, data-str)</i>	Adds a new line to the edit following the line- ptr	No	Yes
<i>Get_Uniq_ID</i>	Returns a unique ID number that identifies this particular edit session.	No	Yes
<i>Set_Csr(line- ptr,col-num [, len-num])</i>	Set the desired location for the cursor following macro execution.	No	Yes

<i>Get_Csr_Col</i>	Returns the column number of the current cursor line	No	Yes
<i>Get_Csr_LPtr</i>	Returns the line pointer of the line containing the cursor	No	Yes
<i>Get_TabNum</i>	Returns the tab number of the currently active tab.	Yes	Yes
<i>Get_TopScrn_LPtr</i>	Returns the line pointer of the line which is currently at the top of screen.	No	Yes
<i>Get_BottomScrn_LPtr</i>	Returns the line pointer of the last data line on the screen.	No	Yes
<i>Get_LeftScrn_Col</i>	Returns the column number of the Left side of the screen	No	Yes
<i>Get_RightScrn_Col</i>	Returns the column number of the Right side of the screen	No	Yes
<i>Set_TopScrn_LPtr</i>	Sets the line pointer which is to appear at the top of screen on macro exit.	No	Yes
<i>Get_Trk_TrkID</i>	Gets the unique Track-ID for a single LPtr	No	Yes
<i>Get_Trk_LPtr</i>	Gets the LPtr for a single Track-ID	No	Yes
<i>Get_Trk_Pos\$</i>	Gets the positioning info from an entry in the Track stack.	No	Yes
<i>Get_Curr_Line\$</i>	Returns the contents of the data line where the cursor was located at the start of macro processing	No	Yes
<i>Get_Curr_Word\$</i>	Returns the contents of the current 'word' where the cursor was located at the start of macro processing	No	Yes
<i>Get_Curr_Path\$</i>	Returns the full path of the Windows current working directory	No	Yes
<i>Get_LBound</i>	Returns the current Left bounds value.	No	Yes
<i>Get_RBound</i>	Returns the current Right bounds value	No	Yes
<i>Get_Modified</i>	Returns the current file modified status as TRUE or FALSE.	No	Yes
<i>Get_Modified_Filename</i>	Returns the modified status for an individual file within an MEdit session	No	Yes
<i>Get_Session_Type\$</i>	Returns a string containing the 'type' of the current edit tab.	No	Yes

	Functions that manage Edit text data	FM	Edit
<code>Get_ANSI2SOURCE_Table\$</code>	Returns a 256 byte string translate table to convert from ANSI to the current SOURCE format.	Yes	Yes
<code>Get_SOURCE2ANSI_Table\$</code>	Returns a 256 byte string translate table to convert from the current SOURCE format to ANSI.		
<code>Get_Line\$(line- ptr)</code>	Returns the current Edit text data for the specified line-ptr	No	Yes
<code>Get_Clr_Line\$(line- ptr)</code>	Returns the current color control line for the specified line-ptr	No	Yes
<code>Get_Line_Len(line- ptr)</code>	Returns the length of the current Edit text data for the specified line-ptr	No	Yes
<code>Get_Line_Type\$(line- ptr)</code>	Returns a text string indicating the particular line type	No	Yes
<code>Get_XLines(line- ptr)</code>	Returns the number of Excluded lines in an Exclude block	No	Yes
<code>Get_XStatus\$(line- ptr)</code>	Returns an X/NX string value indicating the Exclude status of a data line	No	Yes
<code>IS_xxxx(line- ptr)</code>	Tests if a specific line (line-ptr) is a particular line type	No	Yes
<code>Set_Line(line- ptr, data-str)</code>	Will replace the current Edit text data for the specified line-ptr with the new string specified by data-str	No	Yes
<code>Set_Clr_Line(line- ptr, clr-str)</code>	Will replace the current color control string for the specified line-ptr with the new string specified by clr-str	No	Yes
<code>SPF_INS(line- ptr,col-num,data- str)</code>	Will insert <i>StrVar</i> into the text of <i>line-ptr</i> at the indicated column <i>col-num</i>	No	Yes
<code>SPF_REP(line- ptr,col-num,data- str)</code>	Will replace text in <i>line-ptr</i> with <i>StrVar</i> starting at the indicated column <i>col-num</i>	No	Yes
<code>SPF_OVR(line- ptr,col- num,StrVar)</code>	Will overlay text in <i>line-ptr</i> with <i>StrVar</i> starting at the indicated column <i>col-num</i>	No	Yes
<code>SPF_OVR_REP(line- ptr,col- num,data-str)</code>	Will overlay replace text in <i>line-ptr</i> with <i>StrVar</i> starting at the indicated column <i>col-num</i>	No	Yes
<code>Get_Next_LPptr(line- ptr)</code>	Will adjust a line-pointer by moving it either	No	Yes

<i>e_ptr, adjust-amount, line-type)</i>	forward or backward a specified number of lines of a specified line type.		
<i>Get_LPtr(line-ref)</i>	Convert an external line reference to a line pointer	No	Yes
<i>Get_LNum(line-ptr)</i>	Convert a line pointer to an external line number	No	Yes
<i>SPF_Exempt_File(file-ext)</i>	Test if a file extension is in the Text-Exempt list	No	Yes
<i>SPF_Cmd(cmd-str)</i>	Execute an Edit Primary command.	No	Yes
	Functions that manage FIND and LOCATE commands	FM	Edit
<i>Get_LOC_LPtr</i>	Returns the line pointer of the last line found by the LOCATE command	No	Yes
<i>Get_FIND_Firt_LPtr</i>	Returns the line pointer of the FIRST line found by a FIND / CHANGE command	No	Yes
<i>Get_FIND_First_Col</i>	Returns the column number where the FIRST FIND / CHANGE was located	No	Yes
<i>Get_FIND_First_Len</i>	Returns the length of the FIRST FIND / CHANGE search argument	No	Yes
<i>Get_FIND_Last_LPtr</i>	Returns the line pointer of the LAST line found by a FIND / CHANGE command when ALL is coded.	No	Yes
<i>Get_FIND_Last_Col</i>	Returns the column number where the LAST FIND / CHANGE was located when ALL is coded	No	Yes
<i>Get_FIND_Last_Len</i>	Returns the length of the LAST FIND / CHANGE search argument when ALL is coded.	No	Yes
	Functions that manage line command operands	FM	Edit
<i>Get_Handle\$</i>	Returns the .HANDLE for a given data line. Handles are only created by the Get-Handles\$ function.	No	Yes
<i>Drop_Handle\$</i>	Removes a previously assigned Handle	No	Yes
<i>Has_Handle</i>	Returns True / False as to whether a Handle exists for a line.	No	Yes
<i>Get_Label\$</i>	Returns the .LABEL/HANDLE for a given	No	Yes

	data line. If a User created label exists it is selected first. If no User created label, the HANDLE is returned. If neither exist, a Null is returned.		
First_Handle\$ Last_Handle\$	These functions return the current Lowest or Highest issues Handle for the file.	No	Yes
Next_Handle\$ Prev_Handle\$	These functions support looping through the assigned Handles. Each will return the Previous or Next Handle (in Handle issued sequence). A Null ("") is returned when no more exist in the specified direction	No	Yes
Get_Tag\$	Returns the :TAG for a given data line	No	Yes
Get_Src_LCmd\$	Returns a string containing the name of the source line command.	No	Yes
Get_Src1_Lptr	Returns the line pointer of the last line in the source line range, or 0 if not applicable.	No	Yes
Get_Src2_Lptr	Returns the line pointer of the last line in the source line range, or 0 if not applicable.	No	Yes
Get_Src_Op	Returns the optional n value on the source line range.	No	Yes
Get_Src_LMOD\$	Returns the source line command modifiers.	No	Yes
Get_Dest_LCmd\$	Returns a string containing the name of the source line command.	No	Yes
Get_Dest1_Lptr	Returns the line pointer of the first line in the destination line range, or 0 if not applicable.	No	Yes
Get_Dest2_Lptr	Returns the line pointer of the last line in the destination line range, or 0 if not applicable.	No	Yes
Get_Dest_Op	Returns the optional n value on the destination line range command.	No	Yes
Get_Dest_LMOD\$	Returns the destination line command modifiers.	No	Yes
	Functions that retrieve File Manager Overall Data	FM	Edit
FMGet_Mode	Returns the general Mode of the File Manager display.	Yes	No
FMGet_FilePath\$	Returns the contents of the File Path field - EVEN IF the display is not in FilePath mode.	Yes	No
FMGet_FilePatter	Returns the current File Pattern mask -	Yes	No

n\$	EVEN IF the display is not in FilePath mode.		
FMGet_FListName\$	Returns the current FLIST name being displayed. If in FilePath mode, Null is returned.	Yes	No
FMGet_FCount	Returns the current number of line of file data being displayed.	Yes	No
FMGet_Folder_Class({pathname})	Returns the Backup / Normal class of the folder. If a Pathname provided, it returns the status of that folder. If no Pathname, and in FilePath mode, returns the status of the current FilePath.	Yes	No
	Functions that retrieve File Manager Individual File Line Items	Fm	Edit
FMGet_AbbrevName\$(FNum, length)	Returns an abbreviated version of the Filename. Useful when a long filename might cause formatting problems when including the filename in messages. The length parameter allows you to specify the desired length.	Yes	No
FMGet_Attr\$	Returns a short string of 1 character designations representing the file attributes		
FMGet_Backup_Versions(FNum)	Returns the available # of Backup versions for the specified file at line number <i>FNum</i> .	Yes	No
FMGet_Cmd\$(FNum)	Returns the current contents of the Line command area for the specified file at line number <i>FNum</i> .	Yes	No
FMGet_Extension\$(FNum)	Returns the file extension (including the period) for the specified file at line number <i>FNum</i> .	Yes	No
FMGet_FileAttributes(FNum)	Returns the file attribute area for the specified file at line number <i>FNum</i> . This is a binary DWORD of bit flags describing the type of file system entry.	Yes	No
FMGet_File_Class(FNum)	Returns the general file class for the specified file at line number <i>FNum</i> . The Class is one of: a normal file; a Backup of a data file; or a Backup of an associated STATE file.	Yes	No
FMGet_FileName\$(FNum)	Returns the filename for the specified file at line number <i>FNum</i> .	Yes	No

FMGet_FileSize(FNum)	Returns the size of the file specified at line number <i>FNum</i> . The value is returned as a QUAD integer	Yes	No
FMGet_FileSize\$(FNum)	Returns the size of the file specified at line number <i>FNum</i> . The value is returned as a formatted string value. e.g. 9,999,999.	Yes	No
FMGet_FType(FNum)	Returns the type of File Manager entry for the line specified by <i>FNum</i> . e.g. Directory folder (Up or Down), file, FLIST entry etc.	Yes	No
FMGet_Lines(FNum)	Returns the number of Lines in the file (if available) for the file specified at line number <i>FNum</i> . If the Lines value is not available, -1 will be returned	Yes	No
FMGet_Note\$(FNum)	Returns the contents of the Note field (if available) for the file specified at line number <i>FNum</i> . If the Note field is not available, a Null is returned.	Yes	No
FMGet_Path\$(FNum)	Returns the Path to the file (including the trailing \) for the file specified at line number <i>FNum</i> .	Yes	No
FMGet_PRPN\$	Returns the extended property selected by the PRPN column specification		
FMGet_CreationTime\$(FNum)	Returns the Creation time for the file specified at line number <i>FNum</i> . The format returned is: yyyy-mm-dd hh:mm	Yes	No
FMGet_LastWriteTime\$(FNum)	Returns the Last Write time for the file specified at line number <i>FNum</i> . The format returned is: yyyy-mm-dd hh:mm	Yes	No
	Functions that Act on or modify the File Manager Environment	FM	Edit
SPF_FMLCmd(FNum, cmd-string)	Execute a normal FM line command against a specified file at line number <i>FNum</i> . See the Function details for the supported line commands.	Yes	No
FMSet_Cmd(FNum, string)	Modify the line command field for a specified file at line number <i>FNum</i> .	Yes	No
FMSet_Msg(FNum, string)	Issue a message to temporarily overlay the filename for a file line specified by <i>FNum</i> .	Yes	No
Is_Available(fil	Test whether a filename is in use and	Yes	Yes

ename)	unavailable.		
--------	--------------	--	--

Created with the Personal Edition of HelpNDoc: [Ensure High-Quality Documentation with HelpNDoc's Hyperlink and Library Item Reports](#)

Function Details

Introduction

For the descriptions below, each function is shown as it would be called to return a value to a local variable within the script.

All functions which return a string contain a trailing \$ and are shown returning a value to **str-var**, a string variable.

All functions which return a numeric value do not contain a trailing \$ and are shown returning a value to **num-var**, a numeric variable.

Note that in the macro language, there is no requirement that the data returned by a function be assigned to a local script variable before using it. For example, to check if the current Edit file has been modified, it is perfectly correct to code:

```
if Get_Modified = TRUE then ...
```

It does not have to be coded as:

```
LocalVar = Get_Modified  
if LocalVar = TRUE then ...
```

In fact, because the **Get_Modified** function returns a TRUE or FALSE value already, you don't even have to compare it explicitly to TRUE or FALSE. You can use the value returned by the function directly, like this:

```
if Get_Modified then ...
```

This is different - and much simpler - than the IBM Rexx / ISREDIT interface, in which this technique was not available.

To illustrate, here is a simple macro that emulates a built-in command that existed in Tritus SPF:

```
' QMOD.MACRO
IF GET_MODIFIED THEN
    SET_MSG (OK, "DATA MODIFIED")
ELSE
    SET_MSG (OK, "DATA NOT MODIFIED")
END IF
```

With the * on the status line, and color-coded edit tabs, a macro to show the modified state isn't really necessary in SPFLite. It's shown here just as an example.

Detailed description of functions

Note that in the descriptions below, names like *line_ptr* are just symbolic for an actual name or expression you might use. In real code, you can't literally have variable names with minus signs in them (underscores are allowed, though).

Function operands may be either variables or expressions, providing they are of the correct type; you are not limited to just using variables. For a given function, consult the function description as to whether a particular operand should be numeric-valued or string-valued.

Operands whose descriptions end in *-num* or *-ptr* are numeric. Operands whose descriptions end in *-str* are strings.

When the results of a function are not important to you, it is not necessary to assign the results to a variable; you can just call this function as a "procedure call" statement, by simply specifying the function name and its parameters on a single line, with no variable name and = equal sign preceding it.

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Add_Array

<i>num-var</i> = Add_Array (LPtr , array-name)		
Operands:	<i>LPtr</i>	The line pointer after which the new lines from the string array array-name are added. A value of 0 (zero) will mean to add the new line at the end of the current line.
	<i>array-name</i>	The name of the STRING array containing the data line(s) to be inserted next into the Edit file.
Returns:	<i>num-var</i> and <i>RC</i> are both set to 0.	

Created with the Personal Edition of HelpNDoc: [Qt Help documentation made easy](#)

Add_Line

<i>num-var</i> = Add_Line (LPtr , data-string)		
Operands:	<i>LPtr</i>	The line pointer after which the new line containing data-string will be created. A value of 0 (zero) will mean to add the new line at the end of the current line.
	<i>data-string</i>	The data string that is to be inserted next into the Edit file.
Returns:	<i>num-var</i> and <i>RC</i> are both set to 0.	

Created with the Personal Edition of HelpNDoc: [Elevate Your CHM Help Files with HelpNDoc's Advanced Customization Options](#)

Delete_Gbl_Num

<i>num-var</i> = Delete_Gbl_Num ([TblNum] , key-str)		
Operands:	<i>TblNum</i>	Optional. The sub-table number in which the deletion is to be performed. If omitted, a default value of zero is assumed.
	<i>key-str</i>	the key-str of the item to be deleted from the table
Returns:	Returns 0 if successful, or 8 if deletion fails. <i>Msg\$</i> will be "" (if successful, "Error text" if a failure)	

Created with the Personal Edition of HelpNDoc: [Step-by-Step Guide: How to Turn Your Word Document into an eBook](#)

Delete_Gbl_Str

<i>num-var</i> = Delete_Gbl_Str ([TblNum] , key-str)		
--	--	--

Operands:	<i>TblNum</i>	Optional. The sub-table number in which the deletion is to be performed. If omitted, a default value of zero is assumed.
	<i>key-str</i>	the key-str of the item to be deleted from the table
Returns:		Returns 0 if successful, or 8 if deletion fails. Msg\$ will be "" (if successful, "Error text" if a failure)

Created with the Personal Edition of HelpNDoc: [Experience the power of a responsive website for your documentation](#)

Drop_Handle\$

<i>str-var</i> = Drop_Handle\$(Line-ref "ALL")		
Operands:	<i>line-ref</i>	<p>A reference to a data line from which a Line Handle is to be removed. The LineRef can be a numeric Line Pointer, or it can be a string containing a numeric label (" .ABC"), a pseudo-label (" .123"), the string form of a Line Pointer (" !123"), or a Line Handle name (" ._123"). A Tag name(" :ABC") can also be used.</p> <p>If "ALL" is passed as the label-ref, it is a request to release all existing Line Handles.</p>
Returns:		<p>If the LineRef is a valid reference to a data line, and the data line currently has a Line Handle assigned to it, the string form of its current Line Handle (" ._123") is returned, and RC = 0. If the LineRef is valid but the line had no Line Handle, a null string is returned, and RC = 0. If the LineRef is an undefined label or reference to a non-existent data line or to a line that is not a data line, a null string is returned, and RC has a non-zero value.</p> <p>A call of Drop_Handle\$("ALL") returns a null string and RC = 0.</p>
Special Notes		<p>Once a Line Handle is dropped, its value is not used again, unless Drop_Handle\$("ALL") is issued, at which point new Line Handles are assigned starting with handle ._1.</p> <p>The Drop_Handle\$ function supersedes Release_Label\$.</p> <p>All new macros should call Drop_Handle\$.</p>

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

First_Handle\$

<i>str-var</i> = First_Handle\$		
Operands:	None	
Returns:		<p>The lowest Line Handle in existence for the file.</p> <p>If the current file has no Line Handles at all, the function returns a null ("").</p> <p>RC = 0 in all cases.</p>

Special Notes

First_Handle\$ and Last_Handle\$ provide values that can be used in loops. Next_Handle\$ and/or Prev_Handle\$ to loop through and process data line by line. Line Handle order by using a Basic FOR or DO loop.

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

FMGet_AbbrevName\$

```
str-var = FMGet_AbbrevName$(FNum, length)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
	length	The desired length of the abbreviated name

Returns:	STRING	Returns an abbreviated version of the filename. This can be useful at times if the filename is to be included in some message text and long filename could cause parts of the message to be truncated.
-----------------	---------------	--

Created with the Personal Edition of HelpNDoc: [Free help authoring tool](#)

FMGet_Attr\$

```
str-var = FMGet_Attr$(FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
------------------	-------------	---

Returns:	STRING	Returns a string of alphabetic characters which represent the attributes of the file in an easily readable form. Typically this is only 1-4 characters, although there are many possible attributes. The attribute and their single character designation are:
-----------------	---------------	--

N = Normal)
D = Directory)
R = Readonly)
H = Hidden)
S = System)
A = Archive)
T = Temporary)
C = Compressed)
P = Sparse_File)
U = Device)
L = Reparse_Point)
O = Offline)
I = Not_Content_Indexed)
E = Encrypted)
V = Virtual)

Created with the Personal Edition of HelpNDoc: [Transform Your Word Doc into a Professional-Quality eBook with HelpNDoc](#)

FMGet_Backup_Versions

<i>num-var</i> = FMGet_Backup_Versions (FNum)		
Operands:	<i>FNum</i>	The specific line number (of the FM display) to be evaluated.
Returns:	LONG	Returns the current count of available SPFLite Backups for the specified file

Created with the Personal Edition of HelpNDoc: [Eliminate the Struggles of Documentation with a Help Authoring Tool](#)

FMGet_Cmd\$

<i>str-var</i> = FMGet_Cmd\$ (FNum)		
Operands:	<i>FNum</i>	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current contents of the Line Command field for the specified file

Created with the Personal Edition of HelpNDoc: [Easily create EPub books](#)

FMGet_CreationTime\$

<i>str-var</i> = FMGet_CreationTime\$ (FNum)		
Operands:	<i>FNum</i>	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the file creation timestamp in the format: yyyy-mm-dd hh:mm

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

FMGet_Extension\$

<i>str-var</i> = FMGet_Extension\$ (FNum)		
Operands:	<i>FNum</i>	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current file extension (including the . (period) for the specified file

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

FMGet_FCount

<i>num-var</i> = FMGet_FCount		
Operands:	None	

Returns:	LONG	Returns the number of lines in the current File List display.
-----------------	-------------	---

Created with the Personal Edition of HelpNDoc: [Achieve Professional Documentation Results with a Help Authoring Tool](#)

FMGet_FileAttributes

num-var = FMGet_FileAttributes(FNum)																	
Operands:	FNum	The specific line number (of the FM display) to be evaluated.															
Returns:	DWORD	<p>Returns the current file attributes for the specified file. This a DWORD field bits as defined by the Windows File System</p> <p>Windows defines almost 20 various flags, but the ones you would normally about are defined by these equates:</p> <table><tr><td>FM_EQU_READONLY</td><td>=</td><td>&H00000001</td></tr><tr><td>FM_EQU_HIDDEN</td><td>=</td><td>&H00000002</td></tr><tr><td>FM_EQU_SYSTEM</td><td>=</td><td>&H00000004</td></tr><tr><td>FM_EQU_DIRECTORY</td><td>=</td><td>&H00000010</td></tr><tr><td>FM_EQU_ARCHIVE</td><td>=</td><td>&H00000020</td></tr></table> <p>You will notice there is no flag for a 'normal' file. Basically, if a file is not a d it is a 'normal' file.</p>	FM_EQU_READONLY	=	&H00000001	FM_EQU_HIDDEN	=	&H00000002	FM_EQU_SYSTEM	=	&H00000004	FM_EQU_DIRECTORY	=	&H00000010	FM_EQU_ARCHIVE	=	&H00000020
FM_EQU_READONLY	=	&H00000001															
FM_EQU_HIDDEN	=	&H00000002															
FM_EQU_SYSTEM	=	&H00000004															
FM_EQU_DIRECTORY	=	&H00000010															
FM_EQU_ARCHIVE	=	&H00000020															
Special Notes:	<p>When testing, remember that multiple flags may be present at the same time not simply perform a simple comparison of the returned value with one of the equates.</p> <p>Example:</p> <pre>if FMGet_FileAttributes(3) <> FM_EQU_DIRECTORY then</pre> <p>will NOT test properly for a 'normal' file. If it were for example a System folder value would be &H00000014, which is not equal to &H00000010, but it is ce not a normal file.</p> <p>The test should be:</p> <pre>if (FMGet_FileAttributes(3) AND FM_EQU_DIRECTORY) _ <> FM_EQU_DIRECTORY then</pre> <p>The AND operation eliminates all other bits than the Directory bit so that the performed on only that flag.</p>																

Created with the Personal Edition of HelpNDoc: [Eliminate the Struggles of Documentation with a Help Authoring Tool](#)

FMGet_File_Class

num-var = FMGet_File_Class(FNum)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	LONG	Returns the current file class for the specified file. This a LONG field with the following values:

	FM_EQU_BACKUP_DATA FM_EQU_BACKUP_STATE FM_EQU_DATA_FILE
Special Notes:	A file is considered to be a Backup file if it contains a valid SPFLite formatted Date/Time stamp as part of the name, and exists in a \$BACKUP folder. All other files are considered as normal FM_EQU_DATA_FILE .

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Capabilities with a Help Authoring Tool](#)

FMGet_FileName\$

str-var = FMGet_FileName\$ (FNum)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current filename for the specified file.

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Workflow with HelpNDoc's Intuitive UI](#)

FMGet_FilePath\$

str-var = FMGet_FilePath\$		
Operands:	None	
Returns:	STRING	Returns the contents of the File Manager File Path field. This will be returned regardless of the current Mode of the screen. i.e. Even if the display is in, say Recent mode, the value returned is what will be used if the Mode is switched to File Path Mode.

Created with the Personal Edition of HelpNDoc: [Don't Let Unauthorized Users View Your PDFs: Learn How to Set Passwords](#)

FMGet_FilePattern\$

str-var = FMGet_FilePattern\$		
Operands:	None	
Returns:	STRING	Returns the contents of the File Manager File Pattern field. This will be returned regardless of the current Mode of the screen. i.e. Even if the display is in, say Recent mode, the value returned is what will be used if the Mode is switched to File Path Mode.

Created with the Personal Edition of HelpNDoc: [Achieve Professional Documentation Results with a](#)

[Help Authoring Tool](#)

FMGet_FileSize

```
num-var = FMGet_FileSize (FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	QUAD	Returns the current size of the file as a QUAD integer.

Created with the Personal Edition of HelpNDoc: [Easily Add Encryption and Password Protection to Your PDFs](#)

FMGet_FileSize\$

```
str-var = FMGet_FileSize$ (FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current size of the file as a formatted string. e.g. 9,999,999.

Created with the Personal Edition of HelpNDoc: [Streamline Your Documentation Process with HelpNDoc's Project Analyzer](#)

FMGet_FListName\$

```
str-var = FMGet_FListName$
```

Operands:	None	
Returns:	STRING	Returns the contents of the current FLIST name being displayed. If the File Manager is in FilePath mode, a null string is returned.

Created with the Personal Edition of HelpNDoc: [Elevate Your Documentation with HelpNDoc's Project Analyzer Features](#)

FMGet_Folder_Class

```
num-var = FMGet_Folder_Class ({pathname})
```

Operands:	Pathname (Optional)	
Returns:	LONG	Returns the class (type) of folder. The return value will be one of: FM_EQU_NORMAL_FOLDER or FM_EQU_BACKUP_FOLDER A Backup folder is one of the standard SPFLite \$BACKUP folders. If the optional Pathname operand is coded, the answer will be for that specific folder.

If **Pathname** is omitted, then **FM_EQU_BACKUP_FOLDER** will be returned or

- File Manager is in FilePath mode and
- The current FilePath ends in **\\\$BACKUP**

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

FMGet_FType

```
num-var = FMGet_FType(FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
------------------	-------------	---

Returns:	LONG	A File Manager display contains line entries of a variety of types. The value returned by this function indicates the specific type of this line item. Possible are:
-----------------	-------------	--

FM_EQU_DirUp	..\
FM_EQU_DirDown	dirname\
FM_EQU_File	filename
FM_EQU_Path entry	path entry\
FM_EQU_FList entry	FLIST name

Created with the Personal Edition of HelpNDoc: [Make your documentation accessible on any device with HelpNDoc](#)

FMGet_LastWriteTime\$

```
str-var = FMGet_LastWriteTime$(FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
------------------	-------------	---

Returns:	LONG	Returns the Last Write date/Time stamp of the file in the format: yyyy-mm-dd hh:mm
-----------------	-------------	--

Created with the Personal Edition of HelpNDoc: [Achieve Professional Documentation Results with a Help Authoring Tool](#)

FMGet_Lines

```
num-var = FMGet_Lines(FNum)
```

Operands:	FNum	The specific line number (of the FM display) to be evaluated.
------------------	-------------	---

Returns:	LONG	Returns the current number of lines in the file, if available. <i>Lines</i> are available for those files which have a valid STATE file associated with them. If <i>Lines</i> are not available, a value of -1 will be returned
-----------------	-------------	---

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Process with HelpNDoc's Project Analyzer](#)

FMGet_Mode

num-var = FMGet_Mode																				
Operands:	None																			
Returns:	LONG	One of the following values which describe which File Manager display is currently being shown. Note these are defined by built-in thinBasic equates to simplify processing the return value. Possible return values are: <table><tr><td>FM_EQU_FilePath</td><td>The FilePath display</td></tr><tr><td>FM_EQU_Recent</td><td>Recent Files display</td></tr><tr><td>FM_EQU_Found</td><td>Found Files display</td></tr><tr><td>FM_EQU_Opened</td><td>Opened Files display</td></tr><tr><td>FM_EQU_Favorites</td><td>Favorite Files display</td></tr><tr><td>FM_EQU_FLISTS</td><td>FLIST display</td></tr><tr><td>FM_EQU_Paths</td><td>Paths display</td></tr><tr><td>FM_EQU_Config</td><td>Config display</td></tr><tr><td>FM_EQU_Profiles</td><td>" " (Deprecated)</td></tr></table>	FM_EQU_FilePath	The FilePath display	FM_EQU_Recent	Recent Files display	FM_EQU_Found	Found Files display	FM_EQU_Opened	Opened Files display	FM_EQU_Favorites	Favorite Files display	FM_EQU_FLISTS	FLIST display	FM_EQU_Paths	Paths display	FM_EQU_Config	Config display	FM_EQU_Profiles	" " (Deprecated)
FM_EQU_FilePath	The FilePath display																			
FM_EQU_Recent	Recent Files display																			
FM_EQU_Found	Found Files display																			
FM_EQU_Opened	Opened Files display																			
FM_EQU_Favorites	Favorite Files display																			
FM_EQU_FLISTS	FLIST display																			
FM_EQU_Paths	Paths display																			
FM_EQU_Config	Config display																			
FM_EQU_Profiles	" " (Deprecated)																			

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Workflow with HelpNDoc's Intuitive UI](#)

FMGet_Note\$

str-var = FMGet_Note\$(FNum)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current Note string for the file. If none, a Null will be returned.

Created with the Personal Edition of HelpNDoc: [Transform Your Word Doc into a Professional-Quality eBook with HelpNDoc](#)

FMGet_Path\$

str-var = FMGet_Path\$(FNum)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
Returns:	STRING	Returns the current Path, including trailing \, for the file.

Created with the Personal Edition of HelpNDoc: [Make Your PDFs More Secure with Encryption and Password Protection](#)

FMGet_PRPN\$

str-var = FMGet_PRPN\$(FNum) (Also PRP2 through PRP6)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.

Returns:	STRING	Returns the current Extended Property string requested by the selected PR column specification.
-----------------	---------------	---

Created with the Personal Edition of HelpNDoc: [Quickly and Easily Convert Your Word Document to an ePub or Kindle eBook](#)

FMSet_Cmd

num-var = FMSet_Cmd(FNum, string)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
	string	The line command (and its operands) to be stored in the Line Command area of the specified line. A currently entered line command can be canceled by simply replacing the field with a blank, or a new command entered which will be evaluated normally following the execution of this macro.
Returns:	LONG	The command will be executed and any resultant message will appear on the display as usual.

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

FMSet_Msg

num-var = FMSet_Msg(FNum, string)		
Operands:	FNum	The specific line number (of the FM display) to be evaluated.
	string	The text message you wish to appear for this line entry following execution of this macro.

Created with the Personal Edition of HelpNDoc: [Leave the tedious WinHelp HLP to CHM conversion process behind with HelpNDoc](#)

Get_ANSI2SOURCE_Table\$

str-var = Get_ANSI2SOURCE_Table\$		
Operands:		There are no operands.
Returns:		<p>This returns a 256 byte string which is the translate table to convert from the ANSI character set used internally by SPFLite, to the external character set used in the current Profile. e.g. EBCDIC</p> <p>The thinBasic language has the appropriate functions to utilize this table.</p> <p>Note: See also the Get_SOURCE2ANSI_Table\$ function for the companion to this one which will translate characters in the opposite direction.</p>

Get_Arg\$

<i>str-var</i> = Get_Arg\$(<i>arg-num</i>) <i>str-var</i> = Get_Arg\$(<i>first-arg-num</i>,<i>last-arg-num</i>)		
Operands:	<i>arg-num</i>	the number of the only macro operand
	<i>first-arg-num</i>	the number of the first macro operand
	<i>last-arg-num</i>	the number of the last macro operand
Returns:	<p>For (<i>arg-num</i>), it returns the specified macro operand number <i>arg-num</i>, where operands are numbered left to right starting at 1.</p> <p>For (<i>first-arg-num</i>,<i>last-arg-num</i>) it returns a range of operands from operand number <i>first-arg-num</i> through operand number <i>last-arg-num</i>, where the operands are numbered left to right starting at 1.</p> <p>Unless <i>last-arg-num</i> is 0, then <i>first-arg-num</i> must be <= <i>last-arg-num</i>.</p> <p>Macro arguments are treated as simple space delimited operands. If an uncorrected illegal argument number is specified, a "" (null) will be returned.</p> <p>If a valid argument number, RC will be 0 and Msg\$ will be "" (Null) If an invalid argument number, RC will be 8, and Msg\$ will contain an error message</p>	
Special Notes:	If Get_Arg\$(0) is requested, the complete macro arguments are returned as one string, where the individual arguments are separated from each other by blanks.	

Get_Arg_Count

<i>num-var</i> = Get_Arg_Count		
Operands:	none	
Returns:	<p>Returns the number of supplied macro arguments. If no arguments are supplied, 0 will be returned. When the macro header contains one or more default values, the number of default values supplied will be the minimum value returned by Get_Arg_Count.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>	

Get_Arg_KW

<i>num-var</i> = Get_Arg_KW("keyword")
--

Operands:	keyword	The literal value of the keyword to be tested.
Returns:		Returns True or False depending on whether the specified Keyword has been entered as an operand. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:		A full description of using this function will be found in How do I access the command line operands?

Created with the Personal Edition of HelpNDoc: [Don't Let Unauthorized Users View Your PDFs: Learn How to Set Passwords](#)

Get_Arg_KWGroup\$

str-var = Get_Arg_KWGroup\$ ("list-name")		
Operands:	list-name	The literal value of the Keyword list-name for which the value of the entered keyword is to be returned.
Returns:		Returns the actual keyword which was entered on the command operand. If no keyword in the list of valid keywords was entered, this function will return "" (Null) Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:		A full description of using this function will be found in How do I access the command line operands?

Created with the Personal Edition of HelpNDoc: [Easily create Qt Help files](#)

Get_Arg_LRef\$

str-var = Get_Arg_LRef\$ (n)		
Operands:	n	The relative Line Reference value desired, counting from left to right.
Returns:		Returns the specified Line Reference value. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands.

	RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:	A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Bring your WinHelp HLP help files into the present with HelpNDoc's easy CHM conversion](#)

Get_Arg_LRef_Count

<i>num-var = Get_Arg_LRef_Count</i>		
Operands:		None
Returns:		Returns the count of the number of Line Reference type operands entered on the macro operands. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on completion.
Special Notes:		A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Maximize Your PDF Protection with These Simple Steps](#)

Get_Arg_NumLit\$

<i>str-var = Get_Arg_NumLit\$(n)</i>		
Operands:	n	The relative Numeric Literal value desired, counting from left to right.
Returns:		Returns the specified Numeric Literal value. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:		A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Easily create iPhone documentation](#)

Get_Arg_NumLit_Count

<i>num-var = Get_Arg_NumLit_Count</i>		
Operands:		None
Returns:		Returns the count of the number of Numeric Literal type operands entered as macro operands. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on completion.
Special Notes:		A full description of using this function will be found in How do I access the command line out Operands?

Created with the Personal Edition of HelpNDoc: [Effortlessly optimize your documentation website for search engines](#)

Get_Arg_Tag\$

<i>str-var = Get_Arg_Tag\$(n)</i>		
Operands:	n	The relative Tag Operand value desired, counting from left to right.
Returns:		Returns the specified Tag Operand value. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:		A full description of using this function will be found in How do I access the command line out Operands?

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Get_Arg_Tag_Count

<i>num-var = Get_Arg_Tag_Count</i>		
Operands:		None
Returns:		Returns the count of the number of Tag type operands entered as macro operands. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands.

	RC will always be 0 and Msg\$ will be "" (Null) on completion.
Special Notes:	A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Make Documentation a Breeze with a Help Authoring Tool](#)

Get_Arg_TextLit\$

str-var = Get_Arg_TextLit\$(n)		
Operands:	n	The relative Text Literal value desired, counting from left to right.
Returns:		Returns the specified Text Literal value. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on successful completion. RC will be 8 and Msg\$ set to an error message on any error. e.g. an invalid value
Special Notes:		A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Easily share your documentation with the world through a beautiful website](#)

Get_Arg_TextLit_Count

num-var = Get_Arg_TextLit_Count		
Operands:		None
Returns:		Returns the count of the number of Text Literal type operands entered as operands. Note: this function is only valid following successful completion of an SPFLite3 function call to parse the command line operands. RC will always be 0 and Msg\$ will be "" (Null) on completion.
Special Notes:		A full description of using this function will be found in How do I access the out Operands?

Created with the Personal Edition of HelpNDoc: [Make Your PDFs More Secure with Encryption and Password Protection](#)

Get_BottomScrn_LPTr

num-var = Get_BottomScrn_LPTr		
--------------------------------------	--	--

Operands:	none
Returns:	Returns the line pointer of the line which is currently the last Data line displayed on the screen RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Add an Extra Layer of Security to Your PDFs with Encryption](#)

Get_BNDS\$

str-var = Get_BNDS\$		
Operands:	none	
Returns:	Returns a string of nnn THRU nnn MAX e.g 10 THRU 20 or 10 THRU MAX. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Transform Your CHM Help File Creation Process with HelpNDoc](#)

Get_BNDSLIne\$

str-var = Get_BNDSLIne\$		
Operands:	none	
Returns:	Returns NONE or the string containing the current active BNDS line. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Capabilities with HelpNDoc's User-Friendly UI](#)

Get_Clr_Line\$

str-var = Get_Clr_Line\$(line-ptr)		
Operands:	line-ptr	the line pointer for the desired color control line to be fetched.
Returns:	Returns the current color control line data for the specified line-ptr . Line-ptr must be a line pointer for a data line. str-var will be "" (zero-length string) if the line-ptr value is invalid. RC will be 0 and Msg\$ will be "" (Null) for successful completion RC will be 8 and Msg\$ set to an error message on failure (e.g. invalid line-ptr)	

Created with the Personal Edition of HelpNDoc: [Powerful and User-Friendly Help Authoring Tool for Markdown Documents](#)

Get_Csr_Col

<i>num-var</i> = Get_Csr_Col		
Operands:	none	
Returns:	Returns the column number of the current cursor line. If the cursor is not on a valid line in the text area, then 0 (zero) will be returned. RC will always be 0 and Msg\$ will be "" (Null)	
Notes	This value could be saved and used, for example, to return the screen display back to its original position using the Set_Csr_Pos if macro processing has caused it to be altered.	

Created with the Personal Edition of HelpNDoc: [Streamline your documentation process with HelpNDoc's HTML5 template](#)

Get_Csr_LPtr

<i>num-var</i> = Get_Csr_LPtr		
Operands:	none	
Returns:	Returns the line pointer of the line containing the cursor. If the cursor is not on a valid line in the text area, then 0 (zero) will be returned. RC will always be 0 and Msg\$ will be "" (Null)	
Notes	This value could be saved and used, for example, to return the screen display back to its original position using the Set_Csr_Pos if macro processing has caused it to be altered.	

Created with the Personal Edition of HelpNDoc: [Easily create Web Help sites](#)

Get_Curr_Line\$

<i>str-var</i> = Get_Curr_Line\$		
Operands:	none	
Returns:	Returns the contents of the edit data line where the cursor was located at the start of macro processing. If the cursor is not located on a valid data line, value returned will be "" (a zero-length string). RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Effortlessly optimize your documentation website](#)

for search engines

Get_Curr_Path\$

<i>str-var</i> = Get_Curr_Path\$		
Operands:	none	
Returns:	Returns the full path of the Windows 'current' working directory. This is the directory in effect when SPFLite is started up from a command prompt, or the Start directory property when SPFLite is run from an icon, such as from the Windows Desktop. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Don't be left in the past: convert your WinHelp HLP help files to CHM with HelpNDoc](#)

Get_Curr_Word\$

<i>str-var</i> = Get_Curr_Word\$		
Operands:	none	
Returns:	Returns the contents of the current 'word' where the cursor was located at the start of macro processing. If the cursor is not located on a valid data line, or located on white space or delimiters within the line, the returned value will be "" (a zero-length string). Note that 'word' means the same here as it does when used with a FIND 'xxx' WORD command. RC will always be 0 and Msg\$ will be "" (Null)	
Special Notes:	An entire 'word' will be returned regardless of where within the word the cursor is located. It doesn't matter whether the cursor is on the first, last, or any other character of the word. For purposes of determining what a 'word' is, the same rules are used as for the FIND/CHANGE command using the WORD operand. i.e. based on the contents of the Profile WORD character setting.	

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

Get_Dest_LCmd\$

<i>str-var</i> = Get_Dest_LCmd\$		
Operands:	none	
Returns:	The string containing the destination line command if present. This may be an empty (null) string if no destination line range was specified.	

	RC will always be 0 and Msg\$ will be "" (Null)
Special Notes:	Get a string containing the name of the destination line command. This may be A/AA, B/BB, H/HH, W/WW, O/OO, or OR/ORR command, or may be an empty (null) string if no destination line range was specified.

Created with the Personal Edition of HelpNDoc: [Effortlessly Edit and Export Markdown Documents](#)

Get_Dest_Op

num-var = Get_Dest_Op	
Operands:	none
Returns:	Returns the optional numeric line operand, if it is entered, for the destination line command if one is used. If none is entered, this will return 0 (zero). RC will always be 0 and Msg\$ will be "" (Null)
Special Notes:	For example, if you have a destination line range of H5 , Get_Dest_Op will return the number 5 . The function returns 0 when no explicit command operand is specified. For instance, if you define a destination block such as HH which happens to be 5 lines long, the function will still return 0, because the number was not actually specified on the HH block.

Created with the Personal Edition of HelpNDoc: [Make Help Documentation a Breeze with a Help Authoring Tool](#)

Get_Dest_LMod\$

str-var = Get_Dest_LMod\$	
Operands:	none
Returns:	Gets the + or - (post-unexclude or post-exclude) modifier and & (Keep) modifier, if one is present on the destination line range. The returned value is always 2 characters long. Position 1 containing + or - or blank, and position 2 will contain a & or blank. The / and \ line command modifiers are processed, if present, and this gets reflected in first and last LPTR values for the line range, but the / and \ characters themselves are not returned as LMOD values. RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

Get_Dest1_LPTr

num-var = Get_Dest1_LPTr	
Operands:	none

Returns:	Get the line pointer of the first (or only) line in the destination line range, or applicable. The destination line range may be defined by A/AA, B/BB, H/HH, W/WW, or OR/ORR command(s), if present. RC will always be 0 and Msg\$ will be "" (Null)
Special Notes:	Note that for A/AA and B/BB, the value returned is not normalized to be true an A line command. You will have to look at Get_Dest_LCmd to determine command used, and act accordingly.

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

Get_Dest2_LPtr

num-var = Get_Dest2_LPtr	
Operands:	none
Returns:	Get the line pointer of the last (or only) line in the destination line range, or applicable. The destination line range may be defined by A/AA, B/BB, H/HH, W/WW, or OR/ORR command(s), if present. RC will always be 0 and Msg\$ will be "" (Null)
Special Notes:	Note that for A/AA and B/BB, the value returned is not normalized to be true an A line command. You will have to look at Get_Dest_LCmd to determine command used, and act accordingly.

Created with the Personal Edition of HelpNDoc: [How to Protect Your PDFs with Encryption and Passwords](#)

Get_EnvVar\$

str-var = Get_ENVVAR\$ (env-var-str)	
Operands:	env-var-str the name of the environment variable for which the data is to be returned.
Returns:	Returns the string value for the specified var-name. If an unknown variable is specified, a "" (a zero-length string) will be returned. If a valid env-var-str , RC will be 0 and Msg\$ will be "" (Null) If an invalid env-var-str , RC will be 8, and Msg\$ will contain an error message
Special Notes:	For example, Get_ENVVAR\$ ("COMSPEC") would return the name of the system's command processor (Typically C:\Windows\System32\cmd.exe)

Created with the Personal Edition of HelpNDoc: [Achieve Professional Documentation Results with a Help Authoring Tool](#)

Get_EOL_STR\$

<i>str-var</i> = Get_EOL_STR\$ ()		
Operands:	<i>none</i>	There are no operands
Returns:		Returns the string value for the current Profile EOL setting. This does not return the external name of the EOL value (like "CRLF") but the actual EOL string. If the EOL is CRLF it would be X'0D0A'. RC will be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Effortlessly upgrade your WinHelp HLP help files to CHM with HelpNDoc](#)

Get_EXE_Path\$

<i>str-var</i> = Get_EXE_Path\$		
Operands:	<i>none</i>	
Returns:		Returns the full path of the folder where SPFLite.EXE is run. This will usually be C:\Program Files\SPFLite. RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Full-featured EBook editor](#)

Get_FileBase\$

<i>str-var</i> = Get_FileBase\$		
Operands:	<i>none</i>	
Returns:		Returns the base filename of the current edit file. e.g. for MYFILE.TXT it will return MYFILE RC will always be 0 and Msg\$ will be "" (Null)

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

Get_FileDate\$

<i>str-var</i> = Get_FileDate\$		
Operands:	<i>none</i>	
Returns:		Returns the last modified date of the current file in the format YYYY-MM-DD

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Easily share your documentation with the world through a beautiful website](#)

Get_FileName\$

str-var = Get_FileName\$

Operands: none

Returns: Returns the filename of the current edit file. e.g. MYFILE.TXT
RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Get_FileExt\$

str-var = Get_FileExt\$

Operands: none

Returns: Returns the current file extension, including the leading period. e.g. for MYFILE.TXT it would return .TXT
RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

Get_FilePath\$

str-var = Get_FilePath\$

Operands: none

Returns: Returns the current file path for the current file. e.g. C:\Users\Me\Documents\MyFile.txt
RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

Get_FileTime\$

str-var = Get_FileTime\$

Operands: none

Returns:	Returns the last modified time of the current file in the format hh:mm:ss RC will always be 0 and Msg\$ will be "" (Null)
-----------------	--

Created with the Personal Edition of HelpNDoc: [HelpNDoc's Project Analyzer: Incredible documentation assistant](#)

Get_Find_First_Col

num-var = Get_Find_First_Col	
Operands:	none
Returns:	Returns the column number where the FIRST FIND / CHANGE was located the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero). RC will always be 0 and Msg\$ will be "" (Null)
Note:	The former function name Get_Find_Col may still be used for this function but has been deprecated and may be withdrawn in a future release.

Created with the Personal Edition of HelpNDoc: [Maximize Your Productivity with HelpNDoc's Efficient User Interface](#)

Get_Find_First_Len

num-var = Get_Find_First_Len	
Operands:	none
Returns:	Returns the length of the data string found by the FIRST FIND / CHANGE search argument. Note that with Picture and RegEX search operands, this will return a different length than the actual entered argument. Until the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero). RC will always be 0 and Msg\$ will be "" (Null)
Note:	The former function name Get_Find_Len may still be used for this function but has been deprecated and may be withdrawn in a future release.

Created with the Personal Edition of HelpNDoc: [Transform your help documentation into a stunning website](#)

Get_Find_First_LPtr

num-var = Get_FIND_First_LPtr	
Operands:	none
Returns:	Returns the line pointer of the FIRST line found by a FIND / CHANGE command. Until the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero).

RC will always be 0 and Msg\$ will be "" (Null)

Note: The former function name **Get_Find_LPtr** may still be used for this function but has been deprecated and may be withdrawn in a future release.

Created with the Personal Edition of HelpNDoc: [Experience the Power and Simplicity of HelpNDoc's User Interface](#)

Get_Find_Last_Col

num-var = Get_Find_Last_Col

Operands: none

Returns: Returns the column number where the LAST FIND / CHANGE was located. Until the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero).

This will only be available after a FIND/CHANGE command using ALL.

RC will always be 0 and Msg\$ will be "" (Null)

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

Get_Find_Last_Len

num-var = Get_Find_Last_Len

Operands: none

Returns: Returns the length of the **data string** found by the LAST FIND / CHANGE command. Note that with Picture and RegEX search operands, this can be a different length than the actual entered argument. Until the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero).

This will only be available after a FIND/CHANGE command using ALL.

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Get_Find_Last_LPtr

num-var = Get_FIND_Last_LPtr

Operands: none

Returns: Returns the line pointer of the LAST line found by a FIND / CHANGE command. Until the first FIND / CHANGE command is issued for the current Edit file, this will return 0 (zero).

This will only be available after a FIND/CHANGE command using ALL.

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Don't Let Unauthorized Users View Your PDFs: Learn How to Set Passwords](#)

Get_First_LPtr

```
num-var = Get_First_LPtr
```

Operands: none

Returns: Returns a Line Pointer to the first line of the Edit session. If there are no lines in the dataset, **Get_FIRST_LPTR** returns 0 (zero).

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Experience the Power and Ease of Use of a Help Authoring Tool](#)

Get_FullPath\$

```
str-var = Get_FullPath$(file-name)
```

Operands: **file-name** The file name to be fully qualified.

Returns: Returns a fully qualified filename including Drive:\path\filename

If the macro call is issued from a temporary session (CLIP, Set-Edit, (New) Edit, the path returned will be the last displayed path in File Manager.

If the call is issued from a normal Edit/Browse session, the path assumed is that of the currently loaded file.

If the passed *file-name* is Null, or already appears to be qualified, it is returned untouched and the RC will be 8, and Msg\$ will contain an appropriate error message.

Otherwise the RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Make Help Documentation a Breeze with a Help Authoring Tool](#)

Get_Gbl_Num

```
num-var = Get_Gbl_Num([TblNum,]key-str)
```

Operands: **TblNum** Optional. The sub-table number to be searched. If omitted, zero is assumed.

key-str the desired key for which the associated value is desired.

Returns:	Returns the requested value to num-var. If the key-str is invalid / not found (0) is returned. A successful retrieval will set RC to 0 and Msg\$ will be "" (Null) An unsuccessful retrieval, will set RC to 8 and Msg\$ to an error message.
-----------------	--

Created with the Personal Edition of HelpNDoc: [Elevate Your CHM Help Files with HelpNDoc's Advanced Customization Options](#)

Get_Gbl_Num_Count

```
num-var = Get_Gbl_Num_Count ([TblNum])
```

Operands:	TblNum	Optional. The sub-table number for which the count is desired. If omitted, assumed. Zero indicates all items in all sub-tables
Returns:		Returns the current number of stored numeric key/value pairs. RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Experience the power of a responsive website for your documentation](#)

Get_Gbl_Num_Name\$

```
str-var = Get_Gbl_Num_Name$ (index-num)
```

Operands:	index-num	the index number of the desired numeric entry.
Returns:		Returns the current key name associated with this entry to str-var. If successful, RC is always set to 0 and Msg\$ will be "" (Null) If unsuccessful (invalid index-num), RC will be 8, and Msg\$ will contain a message.

Created with the Personal Edition of HelpNDoc: [Maximize Your Productivity with HelpNDoc's Efficient User Interface](#)

Get_Gbl_Num_TableName\$

```
str-var = Get_Gbl_Num_TableName$ (index-num)
```

Operands:	index-num	the index number of the desired numeric entry.
Returns:		Returns the current Table-Number and key name associated with this entry to str-var. The Table-number precedes the key name separated by a comma. e.g. 4 , VARNAME The table number is easily accessed via VAL (str-var) The keyname may be accessed via REMAIN\$ (strvar, ",") If successful, RC is always set to 0 and Msg\$ will be "" (Null)

If unsuccessful (invalid ***index-num***), RC will be 8, and Msg\$ will contain a message.

Created with the Personal Edition of HelpNDoc: [Ensure High-Quality Documentation with HelpNDoc's Hyperlink and Library Item Reports](#)

Get_Gbl_Str\$

```
str-var = Get_Gbl_Str$ ( [TblNum,] key-str )
```

Operands	<i>TblNum</i>	Optional. The sub-table number to be searched. If omitted, zero is assumed.
	<i>key-str</i>	the desired key for which the associated value is desired.
Returns:		Returns the requested value to str-var. If the <i>key-str</i> is invalid / not found, is returned. A successful retrieval will set RC to 0 and Msg\$ will be "" (Null) An unsuccessful retrieval, will set RC to 8 and Msg\$ to an error message.

Created with the Personal Edition of HelpNDoc: [Say Goodbye to Documentation Headaches with a Help Authoring Tool](#)

Get_Gbl_Str_Count

```
num-var = Get_Gbl_Str_Count ( [TblNum] )
```

Operands:	<i>TblNum</i>	Optional. The sub-table number for which the count is desired. If omitted, assumed. Zero indicates all items in all sub-tables
Returns:		Returns the current number of stored numeric key/value pairs. RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Step-by-Step Guide: How to Turn Your Word Document into an eBook](#)

Get_Gbl_Str_Name\$

```
str-var = Get_Gbl_Str_Name$ ( index-num )
```

Operands:	<i>index-num</i>	the index number of the desired numeric entry.
Returns:		Returns the current key name associated with this entry to str-var. If successful, RC is always set to 0 and Msg\$ will be "" (Null) If unsuccessful (invalid <i>index-num</i>), RC will be 8, and Msg\$ will contain a message.

Created with the Personal Edition of HelpNDoc: [Write EPub books for the iPad](#)

Get_Gbl_Str_TableName\$

<i>str-var</i> = Get_Gbl_Str_TableName\$ (<i>index-num</i>)		
Operands:	<i>index-num</i>	the index number of the desired numeric entry.
Returns:	<p>Returns the current Table-Number and key name associated with this entry. The Table-number precedes the key name separated by a comma. e.g. 4 , VARNAME</p> <p>The table number is easily accessed via VAL (str-var) The keyname may be accessed via REMAIN\$ (strvar, ",")</p> <p>If successful, RC is always set to 0 and Msg\$ will be "" (Null) If unsuccessful (invalid <i>index-num</i>), RC will be 8, and Msg\$ will contain an error message.</p>	

Created with the Personal Edition of HelpNDoc: [Don't be left in the past: convert your WinHelp HLP help files to CHM with HelpNDoc](#)

Get_Handle\$

<i>str-var</i> = Get_Handle\$ (<i>line-ref</i>)		
Operands:	<i>line-ref</i>	A reference to a data line to which a Line Handle is to be assigned. The LineRef can be a numeric Line Pointer, or it can be a string containing a normal label (" . ABC "), a pseudo-label (" . 123 ") or the string form of a Line Pointer (" ! 123 "). A Line Handle name (" ._ 123 ") cannot be used, nor can a name (" : ABC ") be used.
Returns:	<p>If the LineRef is a valid reference to a data line, the string form of a Line Handle (" ._123") is returned, and RC = 0. If the LineRef is an undefined label or refers to a non-existent data line or to a line that is not a data line, a null string is returned and RC has a non-zero value.</p> <p>A Line Handle exists independently of a Line Label. A Line Handle assigned to a line is persistent, will exist after the macro which assigns it has terminated, and will be saved in the file's STATE information if STATE SAVING is in effect.</p>	
Special Notes	<p>If a data line already has a Line Handle assigned to it, Get_Handle\$ returns the existing Handle; a new Handle value will not be created. If you wish to force a new Handle value, you must first issue a Drop_Handle\$ for the line, then call Get_Handle\$.</p> <p>The Get_Handle\$ function supersedes Request_Label\$. All new macros should call Get_Handle\$.</p>	

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

Get_INI_Path\$

<i>str-var</i> = Get_INI_Path\$		
Operands:	none	
Returns:	Returns the full path of the SPFLite data storage folder. i.e. the folder where SPFLite data files are stored (Profiles, Macros, etc.) RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Effortlessly upgrade your WinHelp HLP help files to CHM with HelpNDoc](#)

Get_Instance\$

<i>str-var</i> = Get_Instance\$		
Operands:	none	
Returns:	Returns the name of the Instance the current SPFLite session is running. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Full-featured EPub generator](#)

Get_Label\$

<i>str-var</i> = Get_Label\$(<i>line-ptr</i>)		
Operands:	<i>line-ptr</i>	The line pointer of the data line whose Label/Handle is to be obtained.
Returns:	<p>If the <i>line-ptr</i> argument is a valid line pointer to a data line, the function will return the label or handle associated with that data line. If neither exists; it will return a null (empty) string; the RC value will be set to 0 when the <i>line-ptr</i> is associated with a data line.</p> <p>If both a User created label and a Request_Label\$ created handle exist, the user created label is returned. The string value of the returned label will be just what appears in the edit session, with a leading . dot and one to five letters in uppercase case.</p> <p>If no User created label exists, but a Request_Label\$ created handle exists, the Request_Label\$ Handle is returned. The string value will consist of a . dot plus an underscore followed by 'n' digits. e.g. ._123</p> <p>If <i>line-ptr</i> is an invalid line pointer value, or is the line pointer of a non-data line, the function will return a null (empty) string and set RC to 8.</p>	

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

Get_LBound

num-var = Get_LBound		
Operands:	none	
Returns:	Returns the current Left bounds value. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Experience the Power and Simplicity of HelpNDoc's User Interface](#)

Get_Last_LPTr

num-var = Get_Last_LPTr		
Operands:	none	
Returns:	Returns a Line Pointer to the last line of the Edit session. If there are no lines in the dataset, Get_LAST_LPTR returns 0 (zero). RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Write eBooks for the Kindle](#)

Get_LeftScrn_Col

num-var = Get_LeftScrn_Col		
Operands:	none	
Returns:	Returns the column number of the left side of the screen. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Revolutionize Your Documentation Review with HelpNDoc's Project Analyzer](#)

Get_Line\$

str-var = Get_Line\$(<i>line-ptr</i>)		
Operands:	<i>line-ptr</i>	the line pointer for the desired line to be fetched.
Returns:	Returns the current Edit text data for the specified <i>line-ptr</i> . <i>Line-ptr</i> can be a line pointer for a data line or a special line. str-var will be "" (zero-length string) if the <i>line-ptr</i> value is invalid. RC will be 0 and Msg\$ will be "" (Null) for successful completion	

RC will be 8 and Msg\$ set to an error message on failure (e.g. invalid *line*

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Capabilities with HelpNDoc's Project Analyzer](#)

Get_Line_Len

```
num-var = Get_Line_Len(line-ptr)
```

Operands:	<i>line-ptr</i>	the line pointer of the desired line.
-----------	-----------------	---------------------------------------

Returns:	Returns the length of the current Edit data line for the specified <i>line-ptr</i> . <i>line-ptr</i> must be a line pointer for a data line, not for a special line. num-var will be 0 if the <i>line-ptr</i> value is invalid or is not the line pointer of a data line. RC will be 0 and Msg\$ will be "" (Null) for successful completion RC will be 8 and Msg\$ set to an error message on failure (e.g. invalid <i>line</i>
----------	---

Created with the Personal Edition of HelpNDoc: [5 Reasons Why a Help Authoring Tool is Better than Microsoft Word for Documentation](#)

Get_Line_Type\$

```
str-var = Get_Line_Type$(line-ptr)
```

Operands:	<i>line-ptr</i>	the line pointer for which the line's type is to be returned
-----------	-----------------	--

Returns:	Returns a string containing one of the following values: DATA, EXCL, TOP, BOT, TABS, BNDS, COLS, WORD, MARK, MASK, PROF, FILE or NOTE. If the extended NOTE types are used, the returned string will be xNOTE (e.g. ANOTE, BNOTE, XNOTE etc.) If <i>line-ptr</i> is invalid, RC will be set to 8. Otherwise, RC is set to 0 and Msg\$ will be "" (Null)
----------	--

Created with the Personal Edition of HelpNDoc: [Transform your help documentation into a stunning website](#)

Get_LNum

```
num-var = Get_LNum(line-ptr)
```

Operands:	<i>line-ptr</i>	the specific line-ptr for which the line number is desired.
-----------	-----------------	---

Returns:	If the line-ptr is valid, the associated external line number is returned; RC will be 0 and Msg\$ set to "" (null)
----------	--

	If the line-ptr is invalid, "" (null) is returned, RC is set to 8, and Msg\$ is set to an error message.
Special Notes:	<p>Only data lines have line numbers, so if you attempt to use Get_LNum\$ with a line-ptr value for which a call to IS_DATA(line-ptr) would have returned False, a call to GET_LNUM\$(line-ptr) will with RC=8 and a returned value of 0.</p> <p>Assuming the edit session is not operating in Fast Renumber mode, the line number of the last line of the file, and the number of lines in the file, are the same; call this number N. If you call Get_LNUM with an LPTR value <= N+2, the Line Pointer parameter is out of range, and Get_LNUM will return 0.</p>

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

Get_LOC_LPtr

num-var = Get_LOC_LPtr	
Operands:	none
Returns:	<p>Returns the line pointer of the last line found by the LOCATE command. If the last LOCATE was for a special line type (FILE, SPECIAL, etc.) which was not a data line, this will return zero (0). Until the first LOCATE command is issued for the current Edit file, this will return 0 (zero).</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>

Created with the Personal Edition of HelpNDoc: [Effortlessly optimize your documentation website for search engines](#)

Get_LPtr

num-var = Get_LPtr(<i>line-ref</i>)	
Operands:	<p>line-ref a line reference. This could be a line number as it is displayed on the screen, a line label (.HERE, .THERE, etc.)</p> <p>If the line reference is a simple line number, the line-ref value may be specified either as a string expression or as a numeric expression.</p>
Returns:	<p>If the line reference is valid, the associated line pointer is returned; RC is set to 0 and Msg\$ set to "" (null)</p> <p>If the line reference is invalid, 0 (zero) is returned, RC is set to 8, and Msg\$ is set to an error message.</p>

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Get_MacName\$

str-var = Get_MacName\$	
--------------------------------	--

Operands:	none
Returns:	Returns the name of the currently running macro. For a macro called ABC.MACRO, the value of the string returned will be "ABC". RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [iPhone web sites made easy](#)

Get_MarkLine\$

str-var = Get_MARKLine\$	
Operands:	none
Returns:	Returns NONE or the current MARK line data. RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Simplify Your Help Documentation Process with a Help Authoring Tool](#)

Get_MaskLine\$

str-var = Get_MASKLine\$	
Operands:	none
Returns:	Returns NONE or the current MASK line data. RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Why Microsoft Word Isn't Cut Out for Documentation: The Benefits of a Help Authoring Tool](#)

Get_Modified

num-var = Get_Modified	
Operands:	none
Returns:	Returns TRUE or FALSE. True is returned if the current Edit file has been modified since loaded or the last SAVE command; otherwise false if the file is unmodified. RC will always be 0 and Msg\$ will be "" (Null)
Special Notes:	Get_Modified would return True under the same conditions that would produce an asterisk on the Edit status line or would change the color of the Edit tab (assuming the color palettes for modified and unmodified tab colors have been properly set).

In the case of an MEDIT session, Get_Modified will return True if any file in the MEDIT session has been modified.

Created with the Personal Edition of HelpNDoc: [Effortlessly optimize your documentation website for search engines](#)

Get_Modified_FileName

```
num-var = Get_Modified_FileName(filename)
```

Operands:	filename	The name of the file within a MEdit session for which the modified status is required.
------------------	----------	--

Returns:	Returns TRUE or FALSE. True is returned if the specified file has been modified since loaded or the last SAVE command; otherwise false if the file is unmodified. RC will always be 0 and Msg\$ will be "" (Null)
-----------------	--

Special Notes:	If the function is used in a non-MEdit session, it will act exactly like the basic Get_Modified function.
-----------------------	---

Created with the Personal Edition of HelpNDoc: [Make Help Documentation a Breeze with a Help Authoring Tool](#)

Get_Msg\$

```
str-var = Get_Msg$
```

Operands:	none
------------------	------

Returns:	Contains the text of the last message issued by an SPFLite interface function. If a non-successful Return code had been issued by a function, the message associated with that error can be fetched with this function call. Get_Msg\$ and Get_RC do not themselves modify the function-level RC or Message status values, but simply report them. That is, Get_Msg\$ and Get_RC do not have, or set, function-level status values of their own, but only return the last RC or Message value set by some other SPFLite interface function.
-----------------	--

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Get_Next_LPtr

```
num-var = Get_Next_LPtr(line-ptr, adjust-amount, line-type)
```

Operands:	line-ptr	The starting line-pointer to be adjusted and returned.
	col-num	The number of lines by which the line-ptr is to be adjusted. Positive values indicate adjustment downward in the file (towards the last line) and negative values indicate adjustment upward in the file (towards the top line).
	line-type	This is a string operand which may be one of:

	DATA	Adjustment is made where <i>adjust-amount</i> is measured counting only DATA line types
	SPECIAL	Adjustment is made where <i>adjust-amount</i> is measured counting only non-DATA (i.e. SPECIAL) line types
	:tagname	Adjustment is made where <i>adjust-amount</i> is measured counting only lines which have a matching line tag.
Returns:	The adjusted line-ptr as requested. If it is impossible to satisfy the request movement has reached either the top or bottom of the file, depending on direction) then 0 (zero) will be returned and RC and Msg\$ set with appropriate error indications.	
Special Notes:	The Get_Next_LPtr function will typically be used in line scanning loops to move from a currently processed line to the next/previous line of a particular type.	

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Get_Primary_Cmd\$

str-var = Get_Primary_Cmd\$		
Operands:	none	
Returns:	Returns the name of the last entered Primary Command (in uppercase). RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Why Microsoft Word Isn't Cut Out for Documentation: The Benefits of a Help Authoring Tool](#)

Get_Profile\$

str-var = Get_Profile\$(option-str)		
Operands:	option-str	A string containing the name of a PROFILE option. The permitted options are:
	Option name	Return value
	NAME	profile-name
	LOCK	LOCKED UNLOCKED
	ACTION	nnn
	AUTOBKUP	ON OFF
	AUTOCAPS	ON OFF
	AUTOSAVE	ON OFF PROMPT NOPROMPT
	BOM	ON OFF
	BNDLINE	NONE or the current BNDS line
	CAPS	ON OFF AUTO:on AUTO:off
	CASE	C T

	CHANGE	DS CS
	COLLATE	name
	COLS	ON OFF
	DCB	RECFM LRECL and EOL, separated by space
	EMACRO	NONE or the EMACRO string
	EOL	CRLF LF CR NL X'xx'
	HEX	ON OFF
	HIDE	X ON OFF FILE ON OFF
	HILITE	FIND ON OFF AUTO ON OFF
	IMACRO	NONE or the IMACRO string
	LRECL	nnn
	MACLIB	NONE or the MACLIB string
	MARK	ON OFF
	MARKLINE	NONE or the current MARK line
	MASKLINE	NONE or the current MASK line
	MINLEN	nnn
	NOTIFY	NONE EDIT ALL
	PAGE	ON OFF [+/-nn]
	PRESERVE	ON OFF C
	RECFM	U F V VBI VLI
	SCROLL	CSR PAGE HALF ... Etc.
	SOURCE	ANSI UTF8 UTF16 UTF16BE EBCDIC
	START	PRIOR LABEL FIRST LAST NEW
	STATE	ON OFF
	SUBARG	NONE <i>string</i>
	SUBCMD	NONE string
	TABS	ON OFF
	TABSLINE	NONE or the current TABS definition line.
	WORD	A string containing all characters which are considered valid Word characters for the current Profile.
	XFORM	NONE or the XFORM name
	XTABS	nnn
Returns:	The string value of the requested option is returned.	
	RC will be 8 if the <i>option-str</i> is not a valid string PROFILE option name; otherwise	
Special Notes:	There is no corresponding general function to set a PROFILE option. To accomplish this, you must issue an SPF_CMD for the desired setting. For example: SPF_CMD("START FIRST")	
	For numeric settings like LRECL, if you need to work with the returned value in numeric form, you can enclose the request in a VAL() function.	
	Example:	
	DIM MyLRECL AS STRING	
	DIM MyLRECLn AS LONG	

```
MyLRECL = Get_Profile$("LRECL")  
MyLRECLn = VAL(Get_Profile$("LRECL"))
```

Created with the Personal Edition of HelpNDoc: [Converting Word Docs to eBooks Made Easy with HelpNDoc](#)

Get_RBound

num-var = Get_RBound

Operands: none

Returns: Returns the current Right bounds value. If the right bounds is MAX, 0 (zero) is returned.

RC will always be 0

Special Notes: Note that a return value of 0 here means that there is **no** maximum (the right boundary has no arbitrary limit), rather than implying that the right boundary is "column 0".

Created with the Personal Edition of HelpNDoc: [Easily create HTML Help documents](#)

Get_RightScrn_Col

num-var = Get_RightScrn_Col

Operands: none

Returns: Returns the column number of the right side of the screen.

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [News and information about help authoring tools and software](#)

Get_RC

num-var = Get_RC

Operands: none

Returns: The Return Code from the last executed SPFLite function call. The return value will generally be 0, 4 or 8, where 0 = success, 4 = warning and 8 = failure.

The specifics of the error can be obtained via [Get_MSG\\$](#). In many cases, depending on the particular function, the RC may be returned directly from the function call as well. See the specifics for each function call.

Get_Msg\$ and **Get_RC** do **not** themselves modify the function-level RC or Message status values, but simply report them. That is, **Get_Msg\$** and **Get_RC**

do not have, or set, function-level status values of their own, but only return last RC or Message value set by some **other** SPFLite interface function.

Created with the Personal Edition of HelpNDoc: [Revolutionize Your CHM Help File Output with HelpNDoc](#)

Get_Select_First_LPtr

```
num-var = Get_Select_First_LPtr
```

Operands: *None*

Returns: Returns the line-ptr to the first line of a selected text block. If no block is selected, 0 (zero) will be returned.

RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Workflow with HelpNDoc's Intuitive UI](#)

Get_Select_Last_LPtr

```
num-var = Get_Select_Last_LPtr
```

Operands: *None*

Returns: Returns the line-ptr to the last line of a selected text block. If no block is selected, 0 (zero) will be returned.

RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Elevate Your CHM Help Files with HelpNDoc's Advanced Customization Options](#)

Get_Select_Col

```
num-var = Get_Select_Col
```

Operands: *None*

Returns: Returns the left-hand column number of the selected block. If no block is selected, 0 (zero) will be returned.

RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Streamline Your Documentation Process with a Help Authoring Tool](#)

Get_Select_Len

```
num-var = Get_Select_Len
```

Operands:	<i>None</i>
Returns:	Returns the length of the current selected area. If no block is currently selected (zero) will be returned. RC is always set to 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Experience the Power and Ease of Use of HelpNDoc for CHM Help File Generation](#)

Get_Session_Type\$

<i>str-var = Get_Session_Type\$</i>	
Operands:	none
Returns:	Returns a string containing the 'type' of the current edit tab. This will be one of: BROWSE EDIT VIEW CLIP-EDIT SET-EDIT EFT-EDIT MEDIT RDONLY RC will always be 0 and Msg\$ will be "" (Null)

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

Get_SETVAR\$

<i>str-var = Get_SETVAR\$(set-var-str)</i>	
Operands:	<i>set-var-str</i> the name of the SPFLite SET variable for which the data is to be returned
Returns:	Returns the string value for the specified SET variable named by <i>set-var-str</i> . If an unknown variable name is specified, a "" (a zero-length string) will be returned. If a valid <i>set-var-str</i> , RC will be 0 and Msg\$ will be "" (Null) If an invalid <i>set-var-str</i> , RC will be 8, and Msg\$ will contain an error message.
Special Notes:	For example, <i>Get_SETVAR\$ ("ABC")</i> would return the value assigned to the SPFLite SET variable ABC.

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Capabilities with HelpNDoc's User-Friendly UI](#)

Get_SOURCE2ANSI_Table\$

<i>str-var = Get_SOURCE2ANSI_Table\$</i>

Operands:	There are no operands.
Returns:	<p>This returns a 256 byte string which is the translate table to convert from the external character set specified in the Profile SOURCE value (e.g. EBCDIC) used internally by SPFLite (ANSI).</p> <p>The thinBasic language has the appropriate functions to utilize this table.</p> <p>Note: See also the Get_ANSI2SOURCE_Table\$ function for the companion to this one which will translate characters in the opposite direction.</p>

Created with the Personal Edition of HelpNDoc: [Maximize Your Productivity with HelpNDoc's Efficient User Interface](#)

Get_Src_LCmd\$

str-var = Get_Src_LCmd\$	
Operands:	none
Returns:	<p>Contains the line command used to mark the source line range for use by macro. e.g. C/CC, M/MM or the macro name itself if this is a line command.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>
Special Notes:	<p>For line-command macros, this is the macro name (possibly a plural/block) the macro name.</p> <p>For primary-command macros, this may be a C/CC or M/MM command, or an empty (null) string if no C/CC/M/MM was used with a primary-command.</p>

Created with the Personal Edition of HelpNDoc: [Make Your PDFs More Secure with Encryption and Password Protection](#)

Get_Src_Op

num-var = Get_Src_Op	
Operands:	none
Returns:	<p>Contains the optional numeric line operand, if it is entered, for the source line range if used. If none is entered, this will return 0 (zero). For example, if the command entered were C12, the value returned would be 12.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>
Special Notes:	<p>For example, if you have a source line range of C5, Get_Src_Op will return number 5. The function returns 0 when no explicit command operand number is specified. For instance, if you define a source block such as CC which happens to be 5 lines long, the function will still return 0, because the number 5 was not actually specified on the CC block.</p>

Created with the Personal Edition of HelpNDoc: [Effortlessly Edit and Export Markdown Documents](#)

Get_Src_LMOD\$

<i>str-var</i> = Get_Src_LMOD\$		
Operands:	none	
Returns:	<p>The line command modifiers when present.</p> <p>The / and \ line command modifiers are processed, if present, and this gets reflected in first and last LPTR values for the line range, but the / and \ command modifiers themselves are not returned as LMOD values.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>	
Special Notes:	<p>Gets the + or – (post-unexclude or post-exclude) modifier and & (Keep) modifier, if one is present on the source line range. The returned value is always two characters long. Position 1 will contain + or – or blank, and position 2 will contain a & or blank.</p> <p>The / and \ line command modifiers are processed, if present, and this gets reflected in first and last LPTR values for the line range, but the / and \ command modifiers themselves are not returned as LMOD values.</p>	

Created with the Personal Edition of HelpNDoc: [Experience the Power and Simplicity of HelpNDoc's User Interface](#)

Get_Src1_LPtr

<i>num-var</i> = Get_Src1_LPtr		
Operands:	none	
Returns:	<p>Contains the line pointer of the first line in the selected line range marked by the macro.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>	

Created with the Personal Edition of HelpNDoc: [Free EPub producer](#)

Get_Src2_LPtr

<i>num-var</i> = Get_Src2_LPtr		
Operands:	none	
Returns:	<p>Contains the line pointer of the last line in the selected line range marked by the macro. Note: Src1 and src2 may be the same for a single-line selection.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>	

Created with the Personal Edition of HelpNDoc: [Effortlessly Publish Your Word Document as an eBook](#)

Get_TabNum

<i>num-var</i> = Get_TabNum		
Operands:	none	
Returns:	The number of the currently active tab. (File Manager is always 1, the 1st is 2, etc,) RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Streamline your documentation process with HelpNDoc's HTML5 template](#)

Get_TabsLine\$

<i>str-var</i> = Get_TABSLINE\$		
Operands:	none	
Returns:	Returns NONE or the string containing the current active TABS line. RC will always be 0 and Msg\$ will be "" (Null)	

Created with the Personal Edition of HelpNDoc: [Effortlessly Support Your Windows Applications with HelpNDoc's CHM Generation](#)

Get_Tag\$

<i>str-var</i> = Get_Tag\$(<i>line-ptr</i>)		
Operands:	<i>line-ptr</i>	The line pointer of the data line whose line tag is to be obtained.
Returns:	If the <i>line-ptr</i> argument is a valid line pointer to a data line, the function will return the tag associated with that data line, if one exists; otherwise it will return a (empty) string; the RC value will be set to 0 when the line-ptr is associated with a valid data line. The string value of the returned tag will be just as it appears in the edit session with a leading : colon and one to five letters in upper case. If <i>line-ptr</i> is an invalid line pointer value, or is the line pointer of a non-data line, the function will return a null (empty) string and set RC to 8.	

Created with the Personal Edition of HelpNDoc: [Experience the Power and Simplicity of HelpNDoc's User Interface](#)

Get_Trk_Pos\$

<i>str-var</i> = Get_Trk_Pos\$(index-number)								
Operands:	<i>index-number</i>	The index-number indicates which Track position is desired, where: (1) indicates the latest, (2) indicates the prior entry, (3) the next previous, etc. If the index-number is not specified, the default is (1).						
Returns:	<p>The answer returned is a simple comma delimited String of the format</p> <p style="text-align: center;">TOS-Line,Csr_LPtr,CSR_Col</p> <p>Where:</p> <table><tr><td>TOS-Line</td><td>The Lptr of the Top-of-Screen line in character format.</td></tr><tr><td>Csr_LPtr</td><td>The Lptr of the line containing the cursor in character format. If this value is zero, it indicates the cursor is on the command line.</td></tr><tr><td>Csr_Col</td><td>The location of the cursor within the text line in character format. If zero it indicates the cursor is at the beginning of the Line-number area.</td></tr></table> <p>The Return-code should be checked before using the return value, since errors are possible. Currently the two error conditions are:</p> <ul style="list-style-type: none">• The index-number references a Stack entry which has never been used• The Stack entry has been used, but now references a Top-of-Screen line which no longer exists (i.e. it has been deleted)		TOS-Line	The Lptr of the Top-of-Screen line in character format.	Csr_LPtr	The Lptr of the line containing the cursor in character format. If this value is zero, it indicates the cursor is on the command line.	Csr_Col	The location of the cursor within the text line in character format. If zero it indicates the cursor is at the beginning of the Line-number area.
TOS-Line	The Lptr of the Top-of-Screen line in character format.							
Csr_LPtr	The Lptr of the line containing the cursor in character format. If this value is zero, it indicates the cursor is on the command line.							
Csr_Col	The location of the cursor within the text line in character format. If zero it indicates the cursor is at the beginning of the Line-number area.							

Created with the Personal Edition of HelpNDoc: [From Word to ePub or Kindle eBook: A Comprehensive Guide](#)

Get_Trk_LPtr

<i>str-var</i> = Get_Trk_LPtr (track-ID)		
Operands:	<i>track-ID</i>	The operand is a <i>track-ID</i> obtained by a prior Get_Trk_TrkID call.
Returns:	The answer returned is the Lptr for the requested ID. The Return-code should be checked before using the return value, since the <i>track-ID</i> could be for a line which has been deleted.	

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Get_Trk_TrkID

```
str-var = Get_Trk_TrkID(LPtr)
```

Operands:	<i>LPtr</i>	The operand is a normal <i>line-pointer</i> (NOT a Line Number)
Returns:		<p>The answer returned is the Track-ID for the requested LPtr.</p> <p>Every line in an Edit session has a permanently assigned Track ID. This ID is assigned once, and is never changed or removed. Obtaining this ID allows the user to use a macro to obtain a guaranteed pointer to a specific data line.</p> <p>The Return-code should be checked before using the return value, since an invalid <i>line-pointer</i> will simply return a zero.</p>

Created with the Personal Edition of HelpNDoc: [Elevate Your CHM Help Files with HelpNDoc's Advanced Customization Options](#)

Get_TopScrn_LPtr

<i>num-var</i> = Get_TopScrn_LPtr		
Operands:	none	
Returns:		<p>Returns the line pointer of the line which is currently at the top of screen page.</p> <p>RC will always be 0 and Msg\$ will be "" (Null)</p>
Special Notes:		<p>This value could be saved and used, for example, to return the screen display back to it's original position using the Set_Csr_Pos if macro processing has caused it to be altered.</p>

Created with the Personal Edition of HelpNDoc: [Elevate your documentation to new heights with HelpNDoc's built-in SEO](#)

Get_Uniq_ID

<i>num-var</i> = Get_Uniq_ID()		
Operands:	<i>none</i>	There are no operands.
Returns:		<p>Returns a unique ID number for this edit session. The number will be unique to the particular instance of SPFLite. No other Edit/Browse tab will ever duplicate the number. This can be useful for sets of related macros that want to store type data and ensure the data is identified as belonging to a particular editing session.</p>

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

Get_WORDCHAR\$

<i>str-var</i> = Get_WORDCHAR\$		
Operands:	none	
Returns:		Returns a string containing all the characters which are considered valid V

	characters for the active edit session.
	RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

Get_XLINES

num-var = Get_XLINES(line-ptr)		
Operands:	line-ptr	The line pointer of the exclude marker for which the size of the exclude block is desired.
Returns:		<p>If line-ptr is not a valid line, or not an Exclude line marker, then 0 (zero) will be returned, RC will be 8, and Msg\$ set to an appropriate text message.</p> <p>If line-ptr is valid exclude marker, the function will return the number of exclude lines in the group, RC will always be 0 and Msg\$ will be "" (Null)</p>

Created with the Personal Edition of HelpNDoc: [Effortlessly optimize your documentation website for search engines](#)

Get_XSTATUS\$

str-var = Get_XSTATUS\$(line-ptr)		
Operands:	line-ptr	The line pointer of the data line whose exclude status is to be obtained.
Returns:		<p>Returns a string containing either X or NX to indicate the line's exclude status.</p> <p>If line-ptr is not a valid line, then "" (null) will be returned, RC will be 8, and Msg\$ set to an appropriate text message.</p> <p>If line-ptr is valid, then RC will always be 0 and Msg\$ will be "" (Null)</p>

Created with the Personal Edition of HelpNDoc: [Effortlessly bring your documentation online with HelpNDoc](#)

Halt

num-var = Halt([rc-num,] [msg-str] [,msg-str] ...)		
Operands:	[rc-num]	<p>this optional operand provides the numeric return code you wish SPFLite3 to return for the success of this macro invocation. If the operand is omitted, then 0 is assumed.</p> <p>The conventions (and allowable values) for these return codes are:</p> <p>RC=0 for success RC=8 when a failure occurs</p>

		<p>There are predefined symbols for these values. OK is the same as 0; and the same as 8.</p> <p>To avoid issues with future versions of SPFLite, only the values 0, and 8 (symbolic equivalents of OK, and FAIL) should be used as return code values.</p> <p><i>msg-str</i> a string containing the text of the message to be issued. When multiple strings are provided as operands, they will be concatenated together with a single blank space between the operands.</p>
Returns:		Nothing is returned. The macro is terminated by this function and control returns to normal SPFLite operation.
Special Notes:		<p>If RC is not zero, SPFLite will prefix the text with RC=8:</p> <p>Because the RC operand is optional, simple success messages can be issued with only the text string operand. For example:</p> <pre>Halt("Normal completion")</pre> <p>is functionally equivalent to:</p> <pre>Halt(OK, "Normal completion")</pre>

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

Has_Handle

<i>num-var</i> = Has_Handle(line-ref)		
Operands:	line-ref	A reference to a data line. The LineRef can be a numeric Line Pointer, or a string containing a normal label (" .ABC"), a pseudo-label (" .123") or the string form of a Line Pointer (" !123"). A Line Handle name (" ._123") cannot be used, nor can a Tag name(" :ABC") be used.
Returns:		If the LineRef is a valid reference to a data line, and the data line currently has a Line Handle assigned to it, the function returns TRUE, and RC = 0. If the LineRef is valid but the line has no Line Handle, the function returns FALSE, and RC = 1. If the LineRef is an undefined label or refers to a non-existent data line or a line that is not a data line, the function returns FALSE and RC has a non-zero value.
Special Notes		Unlike Get_Handle\$, a call to Has_Handle only returns TRUE or FALSE, it does not create any new Line Handle if one is not currently present on the data line in question.

Created with the Personal Edition of HelpNDoc: [Produce online help for Qt applications](#)

Is_Available

<i>num-var</i> = Is_Available(filename)		
Operands:	filename	The full path and filename to be tested
Returns:		Returns one of the following:

- 0 - The file exists and is available
- 1 - The file does not exist
- 2 - The file exists and is currently open in an SPFLite tab
- 3 - The file exists and is locked by some other Windows process

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [How to Protect Your PDFs with Encryption and Passwords](#)

Is_FM

num-var = Is_FM

Operands: none

Returns: Returns TRUE (1) if the macro was launched in the File Manager tab, If launched in an Edit tab, it returns FALSE

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Leave the tedious WinHelp HLP to CHM conversion process behind with HelpNDoc](#)

Is_Line_Cmd

num-var = Is_Line_Cmd

Operands: none

Returns: Returns TRUE (1) if the macro was launched as a line-command macro, by placing its name into the line sequence-area of one line (if used on a single line) or with an **n** or **/** or **** modifier) or two lines (if used in the block-mode form). Otherwise, FALSE (0) is returned.

RC will always be 0 and Msg\$ will be "" (Null)

Created with the Personal Edition of HelpNDoc: [Transform Your Word Doc into a Professional-Quality eBook with HelpNDoc](#)

Is_Primary_Cmd

num-var = Is_Primary_Cmd

Operands: none

Returns: Returns TRUE (1) if the macro was launched as a line-command macro, by placing its name into the line sequence-area of one line (if used on a single line) or with an **n** or **/** or **** modifier) or two lines (if used in the block-mode form). Otherwise, FALSE (0) is returned.

RC will always be 0 and Msg\$ will be "" (Null)

Is_xxxx

num-var = **IS_xxxx(line-ptr)**

Operands: **line-ptr** the line pointer of the line which is to be tested.

Returns: Returns TRUE if the line at **line-ptr** is the indicated type, or FALSE if it is not. If the **line-ptr** is invalid.

The IS_xxxx function exists in the following forms:

Is_Bnds(line-ptr)	a =BNDS> line
Is_Bottom(line-ptr)	the Bottom of Data line
Is_Cols(line-ptr)	a =COLS> line
Is_Data(line-ptr)	a normal text data line
Is_File(line-ptr)	a =FILE> line
Is_Mark(line-ptr)	a =MARK> line
Is_Mask(line-ptr)	a =MASK> line
Is_Note(line-ptr)	a =NOTE> line
Is_Prof(line-ptr)	a =PROF> line
Is_Tabs(line-ptr)	a =TABS> line
Is_Top(line-ptr)	the Top of Data line
Is_ULine(line-ptr)	a User line
Is_XLine(line-ptr)	an excluded line (usually, but not always a data line)
Is_XMarker(line-ptr)	an excluded line mark (the dashed "placeholder" line)
Is_Word(line-ptr)	a =WORD> line

RC will always be 0 and Msg\$ will be "" (Null)

Last_Handle\$

str-var = **Last_Handle\$**

Operands: None

Returns: The highest Line Handle in existence for the file.

If the current file has no Line Handles at all, the function returns a null ("")
RC = 0 in all cases.

Special Notes First_Handle\$ and Last_Handle\$ provide values that can be used in loops

Next_Handle\$ and/or Prev_Handle\$ to loop through and process data line
Line Handle order by using a Basic FOR or DO loop.

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

Next_Handle\$

```
str-var = Next_Handle$(handle-ID)
```

Operands:	Handle-ID	The current Handle ID of a Data Line.
Returns:	The Next highest Line Handle in existence for the file. If there are no further higher handles, a Null ("") will be returned RC = 0 in all cases.	
Special Notes	First_Handle\$ and Last_Handle\$ provide values that can be used in loops Next_Handle\$ and/or Prev_Handle\$ to loop through and process data line Line Handle order by using a Basic FOR or DO loop.	

Created with the Personal Edition of HelpNDoc: [Effortlessly Convert Your Word Doc to an eBook: A Step-by-Step Guide](#)

Prev_Handle\$

```
str-var = Prev_Handle$(handle-ID)
```

Operands:	Handle-ID	The current Handle ID of a Data Line.
Returns:	The Next lowest Line Handle in existence for the file. If there are no further lower handles, a Null ("") will be returned RC = 0 in all cases.	
Special Notes	First_Handle\$ and Last_Handle\$ provide values that can be used in loops Next_Handle\$ and/or Prev_Handle\$ to loop through and process data line Line Handle order by using a Basic FOR or DO loop.	

Created with the Personal Edition of HelpNDoc: [5 Reasons Why a Help Authoring Tool is Better than Microsoft Word for Documentation](#)

Reset_Gbl_Num

```
num-var = Reset_Gbl_Num ( [TblNum] )
```

Operands:	TblNum	Optional. The sub-table number to be cleared. If NO TblNum operand is entered, ALL sub-tables are cleared. Entering a 0 (Zero) is NOT the same as omitting the operand, it will simply clear sub-table 0.
Returns:	Clears the Gbl_Num storage area and returns True if successful, or False if function fails.	

Created with the Personal Edition of HelpNDoc: [Keep Your Sensitive PDFs Safe with These Easy Security Measures](#)

Reset_Gbl_Str

num-var = Reset_Gbl_Str ([TblNum])		
Operands:	TblNum	Optional. The sub-table number to be cleared. If NO TblNum operand is entered, ALL sub-tables are cleared. Entering a 0 (Zero) is NOT the same as omitting the operand, it will simply clear sub-table 0.
Returns:		Clears the Gbl_Str storage area and returns True if successful, or False if the function fails.

Created with the Personal Edition of HelpNDoc: [Elevate your documentation to new heights with HelpNDoc's built-in SEO](#)

Set_Clr_Line

num-var = Set_Clr_Line(<i>line-ptr</i> , <i>clr-str</i>)		
Operands:	line-ptr	the line pointer for the data line to be replaced
	clr-str	the new color control string to be stored in the line referenced by line-ptr .
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)

Created with the Personal Edition of HelpNDoc: [Revolutionize your documentation process with HelpNDoc's online capabilities](#)

Set_Csr

num-var = Set_Csr(<i>line-ptr</i> , <i>col-num</i> , [<i>len-num</i>])		
Operands:	line-ptr	the line pointer of the line where the cursor is to be placed when the macro terminates.
	col-num	the column number where the cursor should be placed. If col-num is zero, the cursor is placed in the line number field of the given line.
	len-num	If a non-zero len-num is provided, then the data starting at col-num , for len-num characters will be highlighted. When len-num is omitted, it is treated as 0; the cursor will be positioned, but no highlighting will be performed.
Returns:		If line-ptr is valid, RC will be 0 and Msg\$ will be "" (null)
		If line-ptr is invalid, RC will be 8 and Msg\$ will contain an error message.
Special Notes:		Requests SPFLite to move the cursor to the specified line and column number when the macro terminates.

Regardless of where a macro's actions may leave the cursor, the cursor will ALWAYS be placed on the command line when the macro exits.

If you wish to place the cursor elsewhere, you MUST issue a Set_Cur command immediately before the macro ends.

Created with the Personal Edition of HelpNDoc: [Keep Your PDFs Safe from Unauthorized Access with These Security Measures](#)

Set_Gbl_Num

```
num-var = Set_Gbl_Num([TblNum,]key-str, value-num)
```

Operands:	TblNum	Optional. The sub-table number in which to store this value. If omitted, a value of zero is assumed.
	key-str	a string value under which to store the numeric value
	value-num	the numeric value to be associated with key-str .
Returns:		Always returns 0 to num-var and RC. Msg\$ will be "" (Null)
Special Notes:		If a key with the value of key-str already exists, the prior value is replaced with the new value-num . If the key does not already exist, it is created.

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Workflow with HelpNDoc's Intuitive UI](#)

Set_Gbl_Str

```
num-var = Set_Gbl_Str([TblNum,]key-str, value-str)
```

Operands:	TblNum	Optional. The sub-table number in which to store this value. If omitted, a value of zero is assumed.
	key-str	a string value under which to store the numeric value
	value-str	the string value to be associated with key-str .
Returns:		Always returns 0 to num-var and RC. Msg\$ will be "" (Null)
Special Notes:		If a key with the value of key-str already exists, the prior value is replaced with the new value-str . If the key does not already exist, it is created.

Created with the Personal Edition of HelpNDoc: [Make CHM Help File Creation a Breeze with HelpNDoc](#)

Set_Line

```
num-var = Set_Line(line-ptr, data-str)
```

Operands:	<i>line-ptr</i>	the line pointer for the data line to be replaced
	<i>data-str</i>	the new data value to be stored in the line referenced by <i>line-ptr</i> .
Returns:		Both <i>num-var</i> and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both <i>num-var</i> and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)

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

Set_MARK

<i>num-var</i> = Set_Mark (MARK-str)		
Operands:	<i>MARK-str</i>	the new data string to replace the MARK line
Returns:		Both <i>num-var</i> and RC will be 0 and Msg\$ will be "" (Null) for success.

Created with the Personal Edition of HelpNDoc: [Streamline Your Documentation Process with HelpNDoc's Intuitive Interface](#)

Set_MASK

<i>num-var</i> = Set_Mask (MASK-str)		
Operands:	<i>MASK-str</i>	the new data string to replace the MASK line
Returns:		Both <i>num-var</i> and RC will be 0 and Msg\$ will be "" (Null) for success.

Created with the Personal Edition of HelpNDoc: [Experience the Power and Ease of Use of a Help Authoring Tool](#)

Set_Msg

<i>num-var</i> = Set_Msg ([rc-num], msg-str [,msg-str] ...)		
Operands:	<i>[rc-num]</i>	this optional operand provides the numeric return code you wish SPFLite to return for the success of this macro invocation. If the operand is omitted, then 0 is assumed.
		The conventions for these return codes are:
		RC=0 for success RC=8 when a failure occurs
		There are predefined symbols for these values. OK is the same as 0; and FAIL is the same as 8.
		To avoid issues with future versions of SPFLite, only the values 0 and 8 (or their symbolic equivalents of OK and FAIL) should be used as return code values.

	<i>msg-str</i>	a string containing the text of the message to be issued. When multiple strings are provided as operands, they will be concatenated together with a single blank space between the operands.
Returns:		The <i>rc-num</i> value is returned. The value of <i>msg-str</i> is what will be returned to the next Get_Msg\$ call.
Special Notes:		<p>If a message is issued with <i>rc non zero</i>, SPFLite will prefix it with "RC=8: ".</p> <p>Because the RC operand is optional, simple success messages can be issued with only the text string operand. For example:</p> <pre>Set_Msg("Normal completion")</pre> <p>is functionally equivalent to:</p> <pre>Set_Msg(OK, "Normal completion")</pre> <p>It is important to understand that RC and message values passed back by SPFLite functions (like Get_Line\$, for instance) are not automatically passed through to the edit user, even when the RC value is non-zero, signifying a warning or error condition was detected. You must explicitly call Set_Msg yourself if you want a message to be displayed back to the user once the macro completes execution.</p> <p>If Set_Msg is followed by any other SPFLite-provided function which itself returns the RC and message values, the values set by Set_Msg will be overridden. If it is your intent to use Set_Msg to pass back a message to the edit user (you want that appears on the edit screen, the call to Set_Msg should be the last function you call before issuing a HALT statement. (Or use the HALT statement itself to issue the RC and message string)</p>

Created with the Personal Edition of HelpNDoc: [Full-featured Kindle eBooks generator](#)

Set_SETVAR

<i>num-var</i> = Set_SETVAR(<i>set-var-str</i> , <i>value-str</i>)		
Operands:	<i>set-var-str</i>	a string containing the name of the SPFLite SET variable for which the data will be changed.
	<i>value-str</i>	the new value to be assigned to the variable named by <i>set-var-str</i>
Returns:		If successful, RC will be 0 and Msg\$ will be "" (Null) If a failure, RC will be 8, and Msg\$ will contain an error message
Special Notes:		For example, Set_SETVAR("ABC", "New value") would set the value of ABC to the string "New value".

Created with the Personal Edition of HelpNDoc: [Single source CHM, PDF, DOC and HTML Help creation](#)

Set_TABS

<i>num-var</i> = Set_TABS(<i>TABS-str</i>)
--

Operands:	TABS-str	the new data string to replace the TABS line
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for success. If the TABS-str is invalid, the TABS line will be nulled and a RC=8 and appropriate error message returned.

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Output with HelpNDoc's Advanced Project Analyzer](#)

Set_TopScrn_LPtr

num-var = Set_TopScrn_LPtr(line-ptr)		
Operands:	line-ptr	the line pointer for the line to be positioned at the top of screen when the macro exits.
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)
Special Notes:		If your macro uses both Set_TopScrn_LPtr and Set_Csr, the positioning of Set_Csr will override that of the Set_TopScrn_LPtr if the desired cursor position is not within the visible screen as set by the Set_TopScrn_LPtr function. i.e. Set_Csr 'wins'.

Created with the Personal Edition of HelpNDoc: [Effortlessly Spot and Fix Problems in Your Documentation with HelpNDoc's Project Analyzer](#)

Set_WORD

num-var = Set_Word(WORD-str)		
Operands:	WORD-str	the new data string to replace the WORD line
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for success.

Created with the Personal Edition of HelpNDoc: [Revolutionize Your Documentation Output with HelpNDoc's Stunning User Interface](#)

SPF_Cmd

num-var = SPF_Cmd(cmd-str [,cmd-str] ...)		
Operands:	cmd-str	the SPFLite primary command you wish executed. When multiple strings are provided as operands, they will be concatenated together with a single blank space between the operands.
Returns:		RC will be set to the return code issued by the command. This value will be 0 for success.

returned in **num-var**. Msg\$ will be set to whatever message is issued by the command.

Special Notes:

The **cmd-str** expression is passed as a primary command to SPFLite

Within the **cmd-str**, all line references should be entered using the **dotted notation**, e.g. the numeric values should be normal line pointers (**in string format**), preceded by a dot. Line 12 should be specified as **. 12** and not just **12**.

The dotted notation is referred to as "temporary line labels" or "line number pseudo-labels", since this notation is an extension to standard ISPF label

See the section, "*Line Labels: Temporary Line Labels, also known as Line Number Pseudo Labels*" in *Working with Line Labels* in the main Help documentation, for more information.

Because SPFLite primary commands do not deal with line pointers, any command expression string you create must contain line **numbers** and not line **pointers**. If you are working with a line pointer, you must ensure that the line pointer is valid and a pointer to a data line, and then convert that line pointer to a line number; otherwise the command you create will not work. You can use the `Is_Data` function to confirm that a line pointer is valid and points to a data line. `Get_LNUM` is used for the conversion.

For example, if you had a variable **myLptr** containing a valid line pointer value that points to a data line, and you wanted to build a DELETE command to delete that line, an SPF_CMD call to do that would look like this:

```
SPF_CMD("DELETE ." + TSTR$(Get_LNUM(myLptr)))
```

If you wish to use a Line Pointer directly, rather than convert it to a Line Number, you would use the ! exclamation-point notation instead of the . dot notation. The same command above could be specified as

```
SPF_CMD("DELETE !" + TSTR$(myLptr))
```

to achieve the same thing. This ! exclamation-point notation works only in macros, and only for the SPF_CMD function. When you use this technique, you must be certain that the line pointer value used is for a data line and not for a special line. Otherwise, the command passed to SPF_CMD will not be executed, and the function will return RC = 8.

Note: Be aware of the difference between concatenating strings via macro syntax and concatenating strings using the multiple operand support in SPF_Cmd.

Example:

```
SPF_Cmd("DELETE !" + TSTR(myLptr))  
would generate a command of:  
DELETE !5
```

While:

```
SPF_Cmd("DELETE !", TSTR(myLptr))  
would generate a command of:  
DELETE ! 5
```

Note the 2nd version has added a blank separating the SPF_Cmd operand which is **NOT** what is desired.

Created with the Personal Edition of HelpNDoc: [Make the switch to CHM with HelpNDoc's hassle-free WinHelp HLP to CHM conversion tool](#)

SPF_Debug

num-var = SPF_Debug (<i>msg-str</i> [, <i>msg-str</i>] ...)		
Operands:	msg-str	the text string you wish displayed. When multiple strings are provided as operands, they will be concatenated together with a single blank between operands.
Returns:		Always RC = 0 and Msg\$ = "" (null)
Special Notes:		<p>During macro development, the SPF_Debug statement can be used to obtain information related to the macro execution. This can be simple trace messages or displays of internal variables.</p> <p>Examples:</p> <pre>SPF_DEBUG ("Starting line loop") SPF_DEBUG ("Var_b=" & TSTR\$(Var_b))</pre> <p>The SPF_DEBUG statement will open a console window in which to display the information.</p>

Created with the Personal Edition of HelpNDoc: [Don't be left in the past: convert your WinHelp HLP help files to CHM with HelpNDoc](#)

SPF_Exec

num-var = SPF_Exec ([{ <u>SYNC</u> <u>ASYNC</u> } ,] [{ <u>HIDDEN</u> <u>NORMAL</u> } ,] <i>cmd-str</i> [, <i>cmd-str</i>])		
Operands:	<u>SYNC</u> / <u>ASYNC</u>	This optional operand indicates how you want the command to run. If <u>SYNC</u> is coded, control will not be returned to the macro until the command is completed. If <u>ASYNC</u> is coded, the command will be issued and control immediately returned to the macro.
	<u>HIDDEN</u> / <u>NORMAL</u>	This optional operand indicates whether you want the command to run in a visible window or not. If <u>HIDDEN</u> is coded, no window will be seen. (And if error messages are displayed, they will not be visible either) If <u>NORMAL</u> is coded, the command will run in a normally visible command window.
	cmd-str	the command you wish executed. When multiple strings are provided as operands, they will be concatenated together with a single blank between operands.
Returns:		RC will be set to the return code from the executed function. This value will be returned in num-var . Msg\$ will be set to "" (null).

Special Notes:	<p>Both SYNC ASYNC and HIDDEN NORMAL are optional operands, and if omitted, SYNC, NORMAL will be assumed.</p> <p>NOTE: If you want to specify the HIDDEN NORMAL operand, you MUST specify the SYNC ASYNC operand as well, it can not be omitted.</p> <p>The cmd-str string expression is passed as a program to be executed. This function operates similar to the SPF_Shell function, except that the command does not run under the command interpreter (CMD.EXE) but is executed directly. The usual rules and issues about locating the program using the current working directory and the system PATH variable apply to calls to SPF_EXEC.</p> <p>Invoking programs to run under CMD.EXE can require complicated quoting and may require double quoting of operands when things like filenames with embedded blanks are required. With SPF_Exec only simple quoting is required, making the building of the string expression for the command much simpler. The function SPF_Quote may be of assistance in creating properly quoted operand values.</p> <p>Because such commands can perform powerful functions (such as deleting files), extra care should be taken to issue these commands correctly.</p>
----------------	---

Created with the Personal Edition of HelpNDoc: [Transform Your Documentation Workflow with HelpNDoc's Intuitive UI](#)

SPF_Exempt_File

<code>num-var = SPF_Exempt_File(file-ext)</code>		
Operands:	<i>file-ext</i>	This specifies a single file extension. It may contain the leading period or not.
Returns:		RC=0 will be set if the extension is NOT in the 'exempt file' list, RC=8 will be set if the extension IS in the 'exempt file' list.
Special Notes:		<p>SPFLite maintains a list of file extension which are to be considered as non-searchable files. For example this exempts these files from being searched by the FFmpeg (or other Files) command. The list may be altered in the Options - File Manager options.</p> <p>This function allows you to determine if a particular file is contained in this list.</p>

Created with the Personal Edition of HelpNDoc: [Step-by-Step Guide: How to Turn Your Word Document into an eBook](#)

SPF_FMLCmd

<code>num-var = SPF_FMLCmd(FNum, cmd-string)</code>		
---	--	--

Operands:	FNum	The specific line number (of the FM display) to be evaluated.																					
	cmd-string	The line command (and its operands) to be issued on the specified line number. Valid supported Line Commands are: <table><tr><td>A, ADD</td><td>Add to a favorite list</td></tr><tr><td>BACKUP, BACK, BK</td><td>Backup the file</td></tr><tr><td>F, FORGET</td><td>Forget file in this list</td></tr><tr><td>J, JOB, SUBMIT, SUB</td><td>Submit the file</td></tr><tr><td>L, LINES, LINE</td><td>Update the STATE lines info</td></tr><tr><td>N, NORM</td><td>Normalize an FLIST</td></tr><tr><td>P, PRINT</td><td>Print the file</td></tr><tr><td>R, REN, RENAME</td><td>Rename the file</td></tr><tr><td>RS, RST</td><td>Restore a file from backup</td></tr><tr><td>T, TOUCH</td><td>Update the Last Write Date</td></tr><tr><td>W</td><td>OPEN using Windows Shell</td></tr></table>	A, ADD	Add to a favorite list	BACKUP, BACK, BK	Backup the file	F, FORGET	Forget file in this list	J, JOB, SUBMIT, SUB	Submit the file	L, LINES, LINE	Update the STATE lines info	N, NORM	Normalize an FLIST	P, PRINT	Print the file	R, REN, RENAME	Rename the file	RS, RST	Restore a file from backup	T, TOUCH	Update the Last Write Date	W
A, ADD	Add to a favorite list																						
BACKUP, BACK, BK	Backup the file																						
F, FORGET	Forget file in this list																						
J, JOB, SUBMIT, SUB	Submit the file																						
L, LINES, LINE	Update the STATE lines info																						
N, NORM	Normalize an FLIST																						
P, PRINT	Print the file																						
R, REN, RENAME	Rename the file																						
RS, RST	Restore a file from backup																						
T, TOUCH	Update the Last Write Date																						
W	OPEN using Windows Shell																						
Returns:	LONG	The command will be executed and any resultant message will appear on the display as usual.																					

Created with the Personal Edition of HelpNDoc: [Maximize Your CHM Help File Capabilities with HelpNDoc](#)

SPF_INS

num-var = SPF_INS(<i>line-ptr</i> , <i>col-num</i> , <i>data-str</i>)		
Operands:	line-ptr	the line pointer for the data line to be modified
	col-num	the column number at which the str-var contents are to be inserted.
	data-str	the text string to be inserted
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)
Special Notes:		Will insert str-var into the text of line-ptr at the indicated column col-num . text from col-num to the end of line will be shifted right to allow insertion of str . If needed, the line's text will be extended with blanks to col-num - 1 before data-str is copied in.

Created with the Personal Edition of HelpNDoc: [Create HTML Help, DOC, PDF and print manuals from 1 single source](#)

SPF_InvChar\$

str-var = SPF_InvChar\$(<i>data-str</i>)		
Operands:	data-str	a string value to be 'cleaned-up' of unprintable characters.

Returns:	<i>data-str</i> with all invalid characters replaced with the default invalid characters specified in Options => General RC will be 0 and Msg\$ will be "" (Null).
Special Notes:	This function performs the same substitution that is done in normal Edit to remove invalid characters from corrupting the screen display. Because of this the result string should NOT be used as if it were the real string. It should only be used for display purposes where the character substitution would not cause confusion.

Created with the Personal Edition of HelpNDoc: [Make Documentation Review a Breeze with HelpNDoc's Advanced Project Analyzer](#)

SPF_Loop_Check

<i>num-var</i> = SPF_Loop_Check(<i>num-var</i>)		
Operands:	<i>option-num</i>	A numeric value that sets the state of the loop-checking option. This should be TRUE or FALSE depending on setting desired. You may also specify ON or 1 or 0.
Returns:		The function will return the previous setting for Loop Check prior to establishing the new value. RC is set to 0 and Msg\$ = is set to "" (null)
Special Notes:		If your macro is going to perform some long running function, such as interacting with the user by prompting, or any other long-running function, you should use this function to disable the normal SPFLite monitoring of macro execution. This will prevent SPFLite from popping up and informing you that the macro possibly have gone into a loop. Disabling the loop check should not normally be done during macro development and testing, but only after you are confident your macro is running properly. If your macro does go into a loop after loop-checking is disabled, the only recourse is to cancel the entire SPFLite session. When you issue any thinBasic function that causes a wait condition, such as the MsgBox function, SPFLite will detect a loop condition in progress, when in fact nothing is wrong. To avoid such problems, you would disable loop checking before issuing a function call like MsgBox , and re-enable loop checking once the MsgBox has completed.

Created with the Personal Edition of HelpNDoc: [Maximize Your Productivity with HelpNDoc's CHM Help File Creation Features](#)

SPF_OVR

<i>num-var</i> = SPF_OVR(<i>line-ptr</i>, <i>col-num</i>, <i>data-str</i>)		
Operands:	<i>line-ptr</i>	the line pointer for the data line to be overlaid.

	col-num	the column number at which the data-str contents are to be overlaid.
	data-str	the new contents of the data line to be overlaid
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)
Special Notes:		Will overlay text in line-ptr with data-str starting at the indicated column col-num . Other text in the line is unaffected. If needed, the line's text will be extended with blanks to col-num - 1 before data-str is copied in. The overlay operation consists of replacing characters from data-str into the same relative location in the line's text only when there are blanks in the original data. Compare the functionality of SPF_OVR to the O/OO edit line commands.

Created with the Personal Edition of HelpNDoc: [Modernize your help files with HelpNDoc's WinHelp HLP to CHM conversion tool](#)

SPF_OVR_REP

num-var = SPF_OVR_REP(line-ptr, col-num, data-str)		
Operands:	line-ptr	the line pointer for the data line to be overlaid. Line-ptr is a normal Internal pointer.
	col-num	the column number at which the data-str contents are to be overlaid.
	data-str	the new contents of data line to be overlaid
Returns:		Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)
Special Notes:		Will overlay replace text in line-ptr with data-str starting at the indicated column col-num . Other text in the line is unaffected. If needed, the line's text will be extended with blanks to col-num - 1 before data-str is copied in. Overlay replace consists of replacing characters from data-str into the same relative location in the line's text only when the characters in data-str are blank. i.e. similar to the previous SPF_OVR except here non-blanks in data-str take precedent over the original text. In SPF_OVR, non-blanks in the original text take precedent. Compare the functionality of SPF_OVR_REP to the OR/ORR edit line commands.

Created with the Personal Edition of HelpNDoc: [Effortlessly Create High-Quality Documentation with a Help Authoring Tool](#)

SPF_PARSE

<pre>num-var = SPF_PARSE(TextLit-num, NumLit-Num, LinRef-num, TagOp-Num, [keyword-set,] [keyword-set,] ...)</pre>		
Operands:	TextLit-Num	Specifies the number of Text Literals allowed along with optional validation flags (see Special Notes below). If no special options provided, the value specifies the fixed number of this operand value must be entered.
	NumLit-Num	Specifies the number of Numeric Literals allowed along with optional validation flags (see Special Notes below). If no special options provided, the value specifies the fixed number of this operand value must be entered.
	LinRef-Num	Specifies the number of Line References allowed along with optional validation flags (see Special Notes below). If no special options provided, the value specifies the fixed number of this operand value must be entered.
	TagOp-Num	Specifies the number of Tag Operands allowed along with optional validation flags (see Special Notes below). If no special options provided, the value specifies the fixed number of this operand value must be entered.
	Keyword-set	<div>Defines a set of allowable keywords. The format of a Keyword set is: "[list-name:] keyword, keyword, (kwalias, kwalias), keyword, ..." <div><i>list-name</i> a name to refer to this set of keywords, to be used with the SPF_Arg_KWGroup function.</div> <div><i>keyword</i> an allowable keyword. If only one keyword is specified, it is treated as a 'present' or 'not present' keyword type.</div> <div><i>keyword list</i> If multiple keywords are specified, they are treated as a set of mutually exclusive keywords.</div> <div><i>alias list</i> if a keyword entry is a bracketed list of keywords, the bracketed list are treated as aliases of each other. When retrieving information about these keywords, the first keyword in the list most one in the bracketed list is treated as the preferred keyword, the most normalized answer desired.</div></div>

Special Notes:	<p>A full discussion of using SPF_Parse will be found in Full Parse Acc</p> <p>Validation Flags</p> <p>Validation flags, when used, are entered by adding then to the speci numeric value. e.g. If 0 to 3 entries are desired, it would be entered a</p> <p>3 + ARG_VAR</p> <p>if these were tag entries which should be validated, it would be</p> <p>3 + ARG_VAR + ARG_DEF</p> <p>When none of the following flags are used, the number entered beco <u>required</u> number of that particular operand.</p> <p>ARG_VAR When this is specified, Parse will allow from 0 to the sp number to be entered.</p> <p>ARG_OPT When this is specified, Parse will allow either 0 or the e specified number of operands</p> <p>ARG_DEF When this is specified, the Line Reference and Tag Op are validated to ensure they are valid, defined referenc</p>
----------------	---

Created with the Personal Edition of HelpNDoc: [Revolutionize Your Documentation Review with HelpNDoc's Project Analyzer](#)

SPF_Post_Do

<i>str-var = SPF_Post_Do (Do-string)</i>		
Operands:	Do-string	a valid DO string to be executed following the termination of the current m
Returns:		RC will be 0 and Msg\$ will be "" (Null). If the Do-string is not valid, the erro be reported by normal DO processing following macro execution.
Special Notes:		<p>Since macros are not permitted to perform activities which Open or other perform cross-tab operations, SPF_Post_Do provides an ability to bypass of that restriction. The macro could, for example, use SPF_Post_Do to op another file for Edit by issuing the following:</p> <p>SPF_Post_Do("(Home)(EraseEOL)[EDIT filename](Enter)")</p> <p>before terminating.</p> <p>The contents of Do-string are not validated in any way, they are simply pla the queue for normal DO processing to process.</p>

Created with the Personal Edition of HelpNDoc: [Easily create CHM Help documents](#)

SPF_Quote\$

<i>str-var = SPF_Quote\$ (data-str)</i>		
Operands:	data-str	a string value to be enclosed in quotes

Returns:	the value of data-str enclosed in quotes RC will be 0 and Msg\$ will be "" (Null).
Special Notes:	<p>The contents of data-str are examined.</p> <p>If data-str is already in the format of a string enclosed in properly-matched SPFLite quote characters of any type (either " double quotes, ' single quotes, or ` accent quotes), and data-str does not contain inner quote characters as data characters, the contents of data-str expression are returned as-is without modification.</p> <p>If data-str does not contain any " double quote characters as data characters, the function returns the value of data-str preceded and followed by " double quotes.</p> <p>If data-str already contains " double quote characters as data characters, but does not contain any ' single quotes as data characters, the function returns the value of data-str preceded and followed by ' single quotes.</p> <p>If data-str already contains " double quote and ' single quote characters as data characters, but does not contain any ` accent quotes as data characters, the function returns the value of data-str preceded and followed by ` accent quotes.</p>

Created with the Personal Edition of HelpNDoc: [5 Reasons Why a Help Authoring Tool is Better than Microsoft Word for Documentation](#)

SPF_REP

num-var = SPF_REP(line-ptr, col-num, data-str)		
Operands:	line-ptr	the line pointer of the data line to be modified.
	col-num	the column number at which the data-str contents are to be replaced.
	data-str	the replacement text string
Returns:	Both num-var and RC will be 0 and Msg\$ will be "" (Null) for successful completion Both num-var and RC will be 8 and Msg\$ set to an error message on failure (invalid line pointer)	
Special Notes:	<p>Will replace text in line-ptr with data-str starting at the indicated column col-num. Other text in the line is unaffected. If needed, the line's text will be padded with blanks to col-num - 1 before data-str is copied in.</p> <p>The replacement operation consists of replacing characters from data-str starting at the same relative location in the line's original text.</p>	

Created with the Personal Edition of HelpNDoc: [Create help files for the Qt Help Framework](#)

SPF_Shell

<i>num-var</i> = SPF_Shell ([{ <u>SYNC</u> <u>ASync</u> } ,] [{ <u>HIDDEN</u> <u>NORMAL</u> } ,] <i>cmd-str</i> [, <i>cmd-str</i>])		
Operands:	<u>SYNC</u> / <u>ASync</u>	This optional operand indicates how you want the command to run. If <u>SYNC</u> is coded, control will not be returned to the macro until the command is complete. If <u>ASync</u> is coded, the command will be issued and control immediately returned to the macro.
	<u>HIDDEN</u> / <u>NORMAL</u>	This optional operand indicates whether you want the command to run in a hidden window or not. If <u>HIDDEN</u> is coded, no window will be seen. (And if error messages are generated, they will not be visible either) If <u>NORMAL</u> is coded, the command will run in a normally visible command window.
	<i>cmd-str</i>	the command you wish executed by the CMD.EXE shell. When multiple strings are provided as operands, they will be concatenated together with a single space between the operands.
Returns:		RC will be set to the return code from the Shell function. This value will also be returned in <i>num-var</i> . Msg\$ will be set to "" (null).
Special Notes:		<p>Both <u>SYNC</u> <u>ASync</u> and <u>HIDDEN</u> <u>NORMAL</u> are optional operands, and if omitted, <u>SYNC</u>, <u>NORMAL</u> will be assumed.</p> <p>NOTE: If you want to specify the <u>HIDDEN</u> <u>NORMAL</u> operand, you MUST specify the <u>SYNC</u> <u>ASync</u> operand as well, it can not be omitted.</p> <p>The cmd-string expression is passed as a Windows system command to the Windows Command Shell. The command will be performed, and control returned. RC will be set to the Errorlevel / Exit code issued by the specified command. The SPF Quote\$ may be of assistance in creating properly quoted operand values.</p> <p>SPF_Shell can be used for entering Windows shell commands like DEL, MD, RMDIR, etc.</p> <p>Because such commands can be powerful (including commands that may delete files), care should be taken to issue these commands correctly.</p>

Created with the Personal Edition of HelpNDoc: [Benefits of a Help Authoring Tool](#)

SPF_Trace

<i>num-var</i> = SPF_Trace (<i>mode-num</i>)		
Operands:	<i>mode-num</i>	Mode may be ON, OFF or ERROR. When ON, a debugging window will appear that shows a trace record for every SPFLite function called, including the parameters it was passed, its RC and any message text generated. When OFF or ERROR, the debugging information will be recorded only when an SPFLite function is called that does not return an RC of 0.
Returns:		RC will be set 8 if you supply an invalid <i>mode-num</i> operand, otherwise RC will be 0.

Special Notes:	The SPF_Trace call is itself always traced, regardless of the old or new mode in effect.
----------------	---

Created with the Personal Edition of HelpNDoc: [Make Documentation a Breeze with HelpNDoc's Clean and Efficient User Interface](#)

SPF_UnQuote\$

str-var = SPF_UnQuote\$(source-str)		
Operands:	source-str	The string to be 'un-quoted'. UnQuote\$ will remove the outer quotes from if it is indeed quoted. Unquoted source strings are returned unchanged. The outer quotes may be any of the standard SPFLite supported quotes. i Single quotes ' - Double quotes " or Back quotes `
Returns:		RC will always be 0 and Msg\$ = "" (Null)

Created with the Personal Edition of HelpNDoc: [Full-featured Help generator](#)

Working with The Interface

There are many common coding requirements which will be needed by many macros. Below are small common coding techniques showing how these various requirements can be handled. These are **not** full, complete macros but only small code blocks showing one particular aspect.

Fragments include:

- [Accessing macro operands as Line Range operands](#)
- [Accessing all macro operands as a list](#)
- [Modifying data in a text line](#)
- [Setting a final Return Code and Message from a Macro](#)
- [Setting a new cursor location to be used when the macro ends](#)
- [Perform a simple action on all lines in a file](#)
- [Use the FIND command to locate text lines to process](#)
- [Locate text lines to process using non-FIND criteria](#)
- [Insert a New Text line into the File](#)
- [Search for and process all lines with a specific Tag ID](#)
- [Locate 'Problem' lines and add a NOTE line to mark them](#)
- [Using Global storage to communicate between Macros](#)
- [Create a new Line Command](#)
- [File Manager: Scan the File List](#)

Accessing macro operands as Line Range operands

Assume a macro which accepts two operands, a starting and an ending line range.

```
dim fromlptr, tolptr as number
fromlptr = Get_LPtr(Get_Arg$(1))           ' Fetch from
operand
tolptr = Get_LPtr(Get_Arg$(2))             ' Fetch to
operand
if fromlptr = 0 or _                        ' Validate what
```



```

we got
  tolptr    = 0 or _
  fromlptr > tolptr then
    Halt(FAIL, "From / To line pointers missing or invalid")
end if

```

Note: The use of `Get_LPtr` to convert the arguments to internal line pointer format. After this code the `fromlptr` and `tolptr` variables are all ready to be used in further processing code. The **if** statement uses line continuation characters `_` to format the statement as one test per line. (Without the underscore, each **thinBasic** statement ends on the line it starts on. This is a standard BASIC coding convention.)

Macro support also allows a more structured form which automatically fetches command operands. Here's the above sample done in that structured format:

```

' MyMacro.MACRO
SUB MainLine(FromArg as string, ToArg as string)
dim fromlptr, tolptra as number
fromlptr = Get_LPtr(FromArg)           ' Convert the
from operand                          '
tolptr = Get_LPtr(ToArg)               ' Convert the to
operand                               '
if fromlptr = 0 or _                   ' Validate what
we got                                '
  tolptra = 0 or _                     '
  fromlptr > tolptra then               '
    Halt(FAIL, "From / To line pointers missing or invalid")
end if                                 '
END SUB                               ' End of
MainLine SUB

```

And a third method is also available, particularly useful with macros that may have many operands available. See [Full Parse Access](#) for details.

Accessing all macro operands as a list

Assume a macro which accepts any number of operands, and processes each one at a time.

```

dim OpIndex as number
dim Operand as string

if Get_Arg_Count = 0 then halt(FAIL, "No operands are
present")

for OpIndex = 1 to Get_Arg_Count      ' Loop through
operands                              '
  Operand = Get_Arg$(OpIndex)         ' Fetch the
operand                               '
  ... process each Operand value ... ' Process it
next

```

Note: This uses a simple FOR / NEXT loop to fetch and process each operand.

Modifying data in a text line

Assuming the macro has a valid line pointer to a specific data line, there are a variety of methods to change the text.

Replace the entire line

This is done with the **Set_Line** function.

```
Set_Line(line-ptr, replacement-text)
```

Which will completely replace any existing text in the line with the *replacement-text* value

Replace a substring of the text

This is done with the **SPF_REP** function

```
SPF_REP(line-ptr, column, replacement-text)
```

Which will completely replace a portion of the existing text, starting at *column* with the *replacement-text*. If the existing text is less than *column* characters long, it will be first extended with spaces.

Insert a text string into the existing text

This is done with the **SPF_INS** function

```
SPF_INS(line-ptr, column, insert-text)
```

Which will insert a string (*insert-text*) into existing text, following the *column* position. If the existing text is less than *column* characters long, it will be first extended with spaces.

Overlay a text string into the existing text

This is done with the **SPF_OVR** function

```
SPF_OVR(line-ptr, column, overlay-text)
```

The **SPF_OVR** function will overlay a character string within a data line starting at a specified column. Overlay is done by comparing relative characters within the overlay area and then copying characters from the provided string to the data line only if the same relative character in the data line is a blank. This is identical in function to the SPFLite line commands **CC** / **OO**.

Overlay Replace a text string into the existing text

This is done with the **SPF_OVR_REP** function

```
SPF_OVR_REP(line-ptr, column, overlay-text)
```

The `SPF_OVR_REP` function will overlay a character string within a data line starting at a specified column. Overlay replace is similar to Overlay (above) except that Overlay Replace gives priority to the **new** character string. The Overlay is done by comparing relative characters within the overlay area and then copying characters from the provided new string **which are non-blank** to the data line regardless of the contents of the data line. Blanks in the new character string are not copied, thus leaving the original data line characters untouched. This is identical in function to the SPFLite line commands **CC** / **ORR**.

Setting a final Return Code and Message from a Macro

When a macro completes, it is often useful to have SPFLite issue a status message to indicate the relative success or failure of the macro processing. For some macros, this message might be the only desired output (for example, from a word-counting macro). This can be done by either of two very similar functions. `Set_Msg` and `Halt`. The difference is that `Set_Msg` establishes the RC and message values and macro processing continues. `Halt` establishes the values and terminates the macro immediately.

```
Set_Msg("This macro ran successfully")
```

```
Set_Msg(0, "The number of words in the file is: " +  
TSTR$(wordctr))
```

```
Halt(4, "There were no strings found")
```

```
Halt(FAIL, "A required operand is missing")
```

The first operand of `Set_Msg/Halt` is optional, but if provided, and the value is numeric, it is assumed to be the desired return code for the macro, and the second is the message string.

If the first operand is a string operand, it is assumed to be the message text, and a default Return Code value of 0 (zero) is assumed.

Note the first example above uses the default RC format, the other three examples provide specific RC values.

A return code of **0** means the macro was successful; and **8** means that the macro has "failed", based on your definition of what you mean for the macro to fail.

Rather than hard-coding these values, you can specify them symbolically, using **OK** for 0, and **FAIL** for 8.

For non-Zero RC values, SPFLite will format the value and prefix the message text with **RC=nn:** e.g. `RC=8: Missing search string`

For compatibility with future releases of SPFLite, the return code operands you use should only contain 0 or 8, or the symbolic names for them.

Setting a new cursor location to be used when the macro ends (Edit Sessions)

You may optionally set a new cursor location to be used on macro exit. If you do not, the cursor will remain where it was when the macro started **or** where some other SPFLite command positioned it, if SPFLite command(s) were invoked by the macro.

The `Set_Csr` function provides the line pointer, the column number, and an optional length value.

If the column number provided is zero, the cursor will be placed in the line number area of the line.

If the optional length parameter is zero, no text will be highlighted.

If the optional length parameter **is** provided, then the text at the cursor location, for the specified length, will be highlighted.

```

Set_Csr(line_ptr, column, 0)      ' Cursor to line_ptr,
column, with no highlighting

Set_Csr(line_ptr, 0)              ' Cursor to the line number
area of line_ptr

Set_Csr(line_ptr, column, 5)      ' Cursor to line_ptr,
column, with a 5 chars highlighted

```

Perform a simple action on all lines in a file

Assume a portion of macro which wants to perform an action on all lines in the file.

```

dim i as number

for i = Get_First_LPtr to Get_Last_LPtr      ' Loop through
the file
    if Is_Data(i) then                        ' Doing only
data lines
        Set_Line(i, ucase$(Get_Line$(i)))    ' SAMPLE
ACTION: uppercase the text
    end if                                    '
next                                          '

```

Note: This uses a simple FOR / NEXT loop to process each line. An `Is_Data` test ensures we only process text data lines rather than on special lines. The action performed here is just a simple uppercase function on each line's data.

An alternate method using the `Get_Next_LPTr` function could be:

```

dim i as number value 1

i = Get_Next_LPTr(i, 1, "DATA")            ' Adjust i to the
next DATA line
do while istrue i                          ' If not end of
file
    Set_Line(i, ucase$(Get_Line$(i)))        ' SAMPLE ACTION:
uppercase the text
    i = Get_Next_LPTr(i, 1, "DATA")          ' Adjust i to the
next DATA line
loop                                        ' loop-de-loop

```

You can write code to access all lines in a file using a simplified syntax. The **WHEN** option allows the check for the correct type of line, right on the **FOR** statement. It operates just like the code above, without requiring to have an **IF** statement nested inside the **FOR**:

```
dim i as number

for i = Get_First_Lptr to Get_Last_Lptr WHEN Is_Data(i)
    Set_Line(i, ucase$(Get_Line$(i)))      ' SAMPLE ACTION:
uppercase the text
next
```

Note: Thanks to Eros Olmi of *thinBasic* for adding this support to his Basic engine for us.

Use the FIND command to locate text lines to process

To locate specific lines to process using the FIND command use the following code. Assume the string to search for is already set up in the variable `lookfor`.

```
SPF_Cmd("FIND FIRST " & lookfor)                '
Issue 1st FIND cmd
do while Get_RC = 0                               '
while found
    IF other tests needed to 'qualify' this record for
processing
        ... process the found record - - -        '
Process it
    ... The line's text can be obtained with
    ... Get_Line$(Get_Find_LPptr)
    ... The found string will be at the column provided
by
    ... Get_Find_Col
end if                                             '
SPF_Cmd("RFIND")                                  ' Look
some more
loop                                              '
```

Note: The ...process the found record ... code can use the data from `Get_Find_LPptr`, `Get_Find_Col` and `Get_Find_Len` functions since the code is only executed following a successful FIND command (`Get_RC = 0`).

Locate text lines to process using non-FIND criteria

To locate specific lines to process using some other criteria than what the FIND command can provide, use the following code.

```
dim i as number
for i = 1 to Get_Last_LPptr                        '
Let's search
    if is_Data(i) then                             ' Only
data lines
        if ... special criteria test ... then      '
Examine the line
```

```

        ... process the line - - -
    end if
end if
next i

```

Note: The ...process the found record ... code can use the data from `Get_Find_LPptr`, `Get_Find_Col` and `Get_Find_Len` functions since the code is only executed following a successful FIND command (`Get_RC = 0`).

Insert a New Text line into the File

This sample assumes the macro has already determined the line-pointer after which a line is to be inserted. We will assume this is in the variable `curr_Lptr`.

```

    SPF_CMD("LINE N1 !" & TSTR$(curr_Lptr)) ' Insert a line
after curr one

```

```

    Set_Line(curr_Lptr + 1, "New text for the new line")

```

```

    INCR curr_Lptr                                ' Adjust
curr_Lptr for the inserted line

```

Note: The new text line is inserted using the primary command **LINE**. The `Set_Line` function adds the new text, and uses **Curr_Lptr + 1** for the line number, since that will be the new line pointer for the inserted line. The third line, which increments the **Curr_Lptr** value, is what would probably be needed in most macros to adjust the line pointer value for the inserted line. That's because **Curr_Lptr** points to the "current" line, and the line inserted *after* the current line would have a line pointer one greater than the current one. Whether this would actually be needed is of course dependent on the design of the logic flow in the specific macro.

If you wish to use the Line **Number** value directly (rather than a **pointer**), the first line of this example can be specified using the . (period) notation like this:

```

    SPF_CMD("LINE N1 ." & TSTR$(Get_LNUM(curr_Lptr)))

```

Search for and process all lines with a specific Tag ID

This sample locates all lines with a Tag of **:ABC** and processes them.

```

dim i as number value 1
dim tagname as string value ":ABC"

i = Get_Next_LPptr(i, 1, tagname)           ' Adjust i to the
next :ABC tagged line
do while istrue i                           ' If not end of
file
    Set_Line(i, ucase$(Get_Line$(i))        ' SAMPLE ACTION:
uppercase the text
    i = Get_Next_LPptr(i, 1, tagname)        ' Adjust i to the

```

```

next DATA line
loop                                     ' loop-de-loop

```

Locate 'Problem' lines and add a NOTE line to mark them

This sample will locate lines which match some unique criteria, and insert a NOTE line following the line to flag the problems.

```

dim cLPtr as number value 1

while cLPtr < Get_Last_LPtr                ' Loop
through all lines
    if Is_Data(cLPtr) then                  ' Only
do data lines
    if ...special criteria one ... then      ' A
problem line?
        SPF_CMD("LINE NOTE !" + TSTR$(cLPtr))
Insert a NOTE line after this one
        Set_Line(cLPtr + 1, "Check this line - special
criteria one")
        incr cLPtr                          '
Adjust cLPtr for the inserted line
    elseif ...special criteria two ... then  ' A
different problem line?
        SPF_CMD("LINE NOTE !" + TSTR$(cLPtr))
Insert a NOTE line after this one
        Set_Line(cLPtr + 1, "Check this line - special
criteria two")
        incr cLPtr                          '
Adjust cLPtr for the inserted line
    end if                                  ' End
of special tests
end if                                      ' End
of Is_Data tests
    incr cLPtr                              ' Bump
to next line
wend

```

Using Global storage to communicate between Macros

In this sample, Macro A saves the location of the cursor when it is invoked in Global storage. Later, another macro Macro B wishes to return the cursor to the original location stored by Macro A.

Macro A

```

'--- Save where we are for Macro B, use key names
MacALine and MacACol for global storage pool
Set_Gbl_Num("MacALine", Get_LNum(Get_Csr_LPtr))
Save where the cursor is

```

```
Set_Gbl_Num("MacACol", Get_Csr_Col)
```

Macro B

```
dim i, j as number

i = Get_Gbl_Num("MacALine")
Get saved line number
j = Get_Gbl_Num("MacACol")
Get saved column
if i = 0 then halt(FAIL, "No location saved by Macro A")
Set_Csr(Get_LPTr(i), j, 0)
Put cursor back where it was
```

Create a new Line Command

To create a new line command, do the following:

- Choose your command name. It must be short and cannot conflict with any existing line or primary commands.
- Line-command macro names by nature must be short; otherwise they cannot be entered in the sequence area of the edit screen.
- Line-command macros cannot contain digits in the name.
- It is possible to have a line-command macro with a single-letter name. Bear in mind that most of the single letter commands are already taken by SPFLite. The letters remaining available for use as single-letter macro names are **K P Q U V Y** and **Z**. The macro files such macros are stored in would use the normal naming conventions. So, an **K** line-command macro would be stored in the file **K.MACRO**.
- For our example, we'll use **CT** (for Center Text). SPFLite will also automatically recognize **CTT** as the block form of this command, and will process any modifiers that imply a block, such as **n** or a **/** or **** modifier.
- You can also force a line macro of any format to be a Block or single line macro; see "Macro Format and Structure" for details.
- Create the macro name as the "singular form" of **CT.MACRO** (not **CTT.MACRO**)
- There is **no other** setup required to activate this as a line command. The presence of **CT.MACRO** in the \MACROS folder is sufficient.

Example:

```
' CT.MACRO
dim tt, tt2 as string
dim lno1, lno2, width, i as number
  ' Ensure we're called correctly
  if Is_Line_Cmd = FALSE then Halt(fail, "CT/CTT macro was
not issued as a line command")

  '----- Get the line number range
  lno1 = Get_Src1_LPTr
line
  lno2 = Get_Src2_LPTr
line
  width = Get_Line_Op
the centering width
  if width = 0 then width = Get_RBund
```



```

        for i = lno1 to lno2                                ' Loop
through the line range
        if Is_Data(i) then                                  ' Just
Data lines
            tt = trim$(Get_Line$(i))                        ' Get
the trimmed text
            if width = 0 then width = Get_Line_Len(i)      ' Use
line length if no other.
            if len(tt) > width then                          '
Center possible?
                Halt(warn, "One or more lines exceed Center
length")
            else
                tt2 = repeat$((width - len(tt)) / 2, " ") + tt
' Center it
                Set_Line(i, tt2)                            '
Stuff it back
                end if                                       '
            end if                                           '
        next                                                '
        halt                                                ' Done

```

Note: The check of **Is_Line_Cmd** to ensure the macro is invoked as a line command. The line range to be processed is obtained from **Get_Src1_LPtr** and **Get_Src2_LPtr**. For a single line selection, these two values would be identical. A **FOR** / **NEXT** loop is used to process each line. An **Is_Data** test ensures we only process text data lines and not special lines.

File Manager: Scan the File List

Most File Manager macros will at some point want to scan the currently displayed File List. This simple sample shows the basic technique to perform this.

```

' FMSample.macro
'----- Process the File Manager File List
'----- Shows fetching a variety of File related information

if isfalse Is_FM then halt(8, "Not running in File Manager")
if isfalse Is_Primary_Cmd then halt(8, "Not running as
Primary Command")

dim i, j as long
dim t as string

'----- Process the File List
for i = 1 to FMGet_FCount when FMGet_FType(i) = FM_EQU_File
' Select only Files

    t = FMGet_FileName$(i)                                ' Get
the FileName

    t = FMGet_FileSize$(i)                                  ' Get
a formatted string of File Size

    j = FMGet_LastWriteTime$(i)                            ' Get

```

the Last Write time of the file

```
    FMSet_Msg(i, "Murphy was here")           ' Set  
    a message on a File Line
```

```
next i                                         ' End  
of file loop  
halt
```

Note: The checks of **Is_FM** and **Is_Primary_Command** ensure the macro is invoked in the proper environment. A **FOR / NEXT** loop is used to process each line. The **FMGET_FCount** provides the number of items in the list, and the **when FMGet_FType** clause is used to filter out only normal files and ignore folder type entries. The separate lines within the **FOR / NEXT** loop show how to retrieve a variety of data about each file entry. There are many other available retrieval functions, see Function Overview for more details.

Working with Line Handles

A **Line Handle** provides a means for a macro to reference a data line in the current Edit session. This is in addition to Line Labels, Line Numbers and Line Pointers.

Line **Tags** can also be used to refer to data lines, but because Tags can refer to multiple lines, they are very different from Line Labels or Line Numbers. Tags are not considered further here.

What is a Line Handle?

A Line Handle is essentially a unique ID that is used to identify a data line. In theory, any data line can have a Line Handle. In practice, only lines that are "of interest" to a macro writer will have Line Handles. The presence of a Line Handle on a line is **optional**.

Why use Line Handles?

When a macro uses functions that refer to data lines, nearly all of those functions take **Line Pointers** as arguments. While Line Pointers work relatively well, they have certain shortcomings:

- Line Pointers can become **invalidated**. This can happen if any lines are inserted or deleted before the line pointed-to by the line pointer. That includes not just data lines but NOTE and PROFILE lines, and lines which are excluded or unexcluded. So, a macro writer must be very careful to avoid any logic that would invalidate a Line Pointer. **Invalidation** is the main drawback to Line Pointers.
- Line Pointers are not "persistent" like labels are. The only way for a macro to re-establish a Line Pointer is for it to find the associated data line again through some kind of search logic. So, any work previously done to find such lines has to be repeated each time.

What about Request/Release?

Macro users may be familiar with the former functions **Request_Label\$** and **Release_Label\$**. Can't these be used as handles? Yes, up to a point, but their implementation had certain limitations:

- Labels used with Request/Release are temporary, and are deleted as soon as the macro creating them terminates, **even** if the macro did not explicitly request for them to be released. So, there is no opportunity to save them as persistent STATE information.
- Request/Release labels are stored where normal labels are. (Users don't notice that currently, because temporary labels disappear as soon as a macro ends.) So, if these were made "persistent", they would become visible on the Edit screen, where they would be a distraction to users with their unusual appearance.
- Because of how they are stored, a data line can have either a normal label or a Request/Release label, but not both. So, these labels are used inconsistently.

The current **Request_Label\$** and **Release_Label\$** functions will be deprecated when Line Handles are implemented. It is recommended that you replace **Request_Label\$** with **Get_Handle\$**, and replace **Release_Label\$** with **Drop_Handle\$** in your macro code. The new functions are not totally identical to the old ones, but are close enough to be used as substitutes in

old macros. Eventually, these old function names will be removed. All new Handle-related functions will contain the word **Handle** in their function name.

For now, **Request_Label\$** will be treated as an alias name for **Get_Handle\$**, and **Release_Label\$** will be treated as an alias name for **Drop_Handle\$**. Even though the old names will continue to exist for a while, they will be just alternate names for the new Handle-related function calls.

Internal properties of Line Handles

- Only data line can have line handles. (Compare to Line Pointers, which can point to anything visible on the Edit screen, like Profile lines, NOTES, etc.)
- When requested by a macro function, a data line is assigned a Line Handle value. This value is a unique ID, no other line will have the same ID.
- The **Line Handle ID** is a string value that can be used in building SPFLite commands in all the same ways as a Line Label. .
- Line Handles are created in the order they are assigned, not in the numeric order of the underlying data lines.
- A Line Handle is a separate attribute of a data line, just like labels, tags and User marks are.
- Although Line Handles can be used like a normal Line Label, they exist independently, so a data line can have a Label, a Handle, **or both**.
- A Line Handle is part of the file's STATE information. If STATE ON is in effect, Line Handle information is saved and is persistent between Edit sessions, like other STATE information is.
- It is possible to remove a Line Handle from a line with the function Drop_Handle\$. If that same data line is assigned a Line Handle again via the function Get_Handle\$, the **new** Handle that is assigned is the "next available" Handle value. The old Handle is not restored, but is permanently discontinued for this Edit file.
- You can remove all Handles from an entire file via Drop_Handle\$("ALL").

External properties of Line Handles

Line Handles essentially behave as if they were a second Line Label on a data line. Since Line Labels are used in Primary Edit Commands, they have these external properties:

- A Line Handle can be used anywhere a Line Label can be used. They may be thought of as a specialized kind of Label, similar to how "pseudo-label" line numbers like **123** are used.
- The main use of Line Handles on primary commands will be those primary commands that are issued by a macro using a call to **SPF_CMD()**.
- The primary command **RESET** will be extended to support RESET HANDLE. RESET LABEL will **not** clear any Line Handles. That protects any existing handles from being accidentally cleared by a RESET LABEL command.
- The primary command **LOCATE** will be extended to support LOCATE HANDLE. LOCATE LABEL will **not** locate lines with Line Handles. When LOCATE HANDLE is issued, an information message will be displayed in the usual message-location to identify the Handle that was found, since Handles are not otherwise visible on the screen nor on the status line.

Behaviour of Line Handles when multiple files are involved

If you edit two files that have saved STATE information, it is possible that Labels and/or Handles

in one file are duplicated in another. If you edited such files together, those duplication's could be confusing. To avoid this problem, SPFLite takes the following actions:

1. If you are editing a single file with Line Labels and/or Line Handles as STATE information, these are accepted as a normal part of the data file. If you COPY a second file with Line Labels and/or Line Handles, any Labels or Handles in the second file are discarded.
2. In a Multi-Edit session, all existing Line Labels and Handles are discarded. You can run macros that utilize Handles, but these will have to be reestablished after the MEdit session is started. (In other words, an MEdit session begins as though a function call of Drop_Handle("ALL") had been issued.)

Note: If the files included in an MEdit session have Labels and/or Handles, **and** the file is **not** saved from the MEdit session, then the handles for the individual file will not be disturbed.

If you wish to ensure that existing Labels and Handles are not lost during an MEdit session, you need to treat the session as though it were a Browse session, and when finished, be sure to issue a CANCEL command.

Created with the Personal Edition of HelpNDoc: [What is a Help Authoring tool?](#)

Sample Macros

```
' CT.MACRO
' Center text on a line. Uses RBound (if active) or specified length
' Syntax: CTnn or CTTnn / CTTnn as line commands      nn is the width to center on
'
' Author: George Deluca
'
dim tt, tt2 as string
dim lno1, lno2, width, i as number
if is_primary_Cmd then halt(fail, "CT macro was not invoked as a line command")

'----- Get the line numbers
lno1 = Get_Src1_Lptr           ' From line
lno2 = Get_Src2_Lptr           ' To line
width = Get_Src_Op             ' Get the centering width
if width = 0 then width = Get_RBound

for i = lno1 to lno2           ' Loop through the line range
  if Is_Data(i) then           ' Just Data lines
    tt = trim$(Get_Line$(i))    ' Get the trimmed text
    if width = 0 then width = Get_Line_Len(i) ' Use line length if no other
    if len(tt) > width then      ' Center possible?
      Set_Msg(8, "One or more lines exceed Center length")
    else
      tt2 = repeat$(width - len(tt) / 2, " ") + tt ' Center it
      Set_Line(i, tt2)           ' Stuff it back
    end if
  end if
next
halt                           ' Done
```

```
-----
' ISRBox.MACRO
' ISRBOX - Draw box with its U/L corner at the cursor
' Syntax: ISRBOX               With cursor sitting on the
'                               desired location. Normally the
'                               command would be assigned to a
'                               command key for convenience.
'
' Author: George Deluca
' Original in Rexx in the IBM macros documentation
```

```

'
dim i, j as number
if Get_Csr_LPtr = 0 then halt(FAIL, "Cursor is not within the text area")'

j = Get_Csr_LPtr                                ' Point at 1st line
for i = 1 to 5                                  ' Set # lines to do
    if j = Get_Last_LPtr then exit for          ' Just in case we hit bottom
    if Is_Data(j) then                          ' Only data lines
        if i = 1 or i = 5 then                  ' If Top or bottom of box
            SPF_Ovr(j, Get_Csr_Col, "+-----+") ' Overlay the box chars
        else                                    ' Else middle lines
            SPF_Ovr(j, Get_Csr_Col, "|         |") ' Overlay the box chars
        end if
        if i = 3 then Set_Csr(j, Get_Csr_Col + 2, 0) ' Put cursor on middle line
    else
        Decr i                                  ' Don't move box row number
    end if
    Incr j                                       ' On to next edit line
next
halt                                           ' Done

```

```

-----
----
' ISRCOUNT.MACRO
'
' ISRCOUNT counts the number of occurrences of a string, and
' issues a message.
'
' Syntax: ISRCOUNT string
'
' Author: George Deluca
' Original by IBM in the Macros documentation

```

```

dim msg as string
if Get_Arg$(0) = "" then halt(fail, "Missing search argument") ' Better have an
operand

SPF_Cmd("FIND ALL " + SPF_Quote$(Get_Arg$(0))) ' Issue a FIND ALL command
if Get_RC <> 0 then halt(fail, "No occurrences of: " + Get_Arg$(1)) ' Tell of error

msg = Get_Msg$                                ' Get the SPFLite message text
' Issue our own format message
halt("ISRCOUNT found", Get_Arg$(0), parse$(msg, " ", 4), "times")

```

```

-----
----
' ISRMASK.MACRO
'
' ISRMASK - Overlay a line with data from the mask line.
' Use either line command O/OO or OR/ORR to specify
' which lines to overlay. O/OO causes nondestructive
' overlay, and OR/ORR causes a destructive overlay.
'
' Author: George Deluca
' Original by IBM in the Macros Documentation

```

```

dim mask as string
dim i as number
spf_debug(Get_Dest_LCmd$)
if left$(Get_Dest_LCmd$, 1) <> "O" then halt(fail, "No overlay range has been
selected")

mask = Get_Profile$("MASK")                    ' Fetch the MASK data

for i = Get_Dest1_LPtr to Get_Dest2_LPtr        ' Loop through line range
    if Is_Data(i) then                          ' Only Data lines
        if Get_Dest_LCmd$ = "O" or Get_Dest_LCmd$ = "OO" then ' Normal O/OO type
            overlay?
        end if
    end if
next

```

```

        SPF_Ovr(i, 1, mask)           ' Overlay the mask
    else                             ' Must be OR/ORR type
        SPF_Ovr_Rep(i, 1, mask)      ' Force Overlay the mask
    end if                           '
end if                               ' End IsData
next                                 '
halt                                ' Done

```

```

-----
' PB.MACRO
' Run a batch compile of the current program. If the compiler reports an error,
' read the LOG file to locate the error, and move the cursor to the line in error
' and display the compiler's error message. This demonstrates how to read and
' process an external file using the thinBasic FILE module.
'
' Syntax: PB
'
' Author: George Deluca
'
dim cmd, errmsg, tt as string
dim errlin, errcol, i as number
dim fHandle as DWORD
uses "FILE"                               ' Attach the FILE module

'----- Save unmodified files
SPF_Cmd("SAVEALL COND")

'----- Build command line to run the compiler
cmd = $DQ + "D:\Google Drive\Misc Data\PBWin10.bat" + $DQ + " "
cmd += $DQ + mid$(Get_FilePath$, 3) + $DQ + " "
cmd += $DQ + Get_FileName$ + $DQ

'----- Do the compile and get the RC result
SPF_EXEC(cmd)
if Get_RC = 0 then halt("PB10 compile successful")

'----- Handle the compiler error, get the LOG file
fHandle = FILE_OPEN(Get_FilePath$ + "\" + Get_FileBase$ + ".LOG" , "INPUT")
if fHandle = 0 then halt(fail, "PB10 compile failed, can't open LOG file")
for i = 1 to 6                             ' Read line 6 of the LOG
    errmsg = FILE_LineInput(fHandle)
next i
i = FILE_Close(fHandle)

'----- Extract error line number, column and error message text
'----- Sample error line:  Error 442 in C:\Documents\Source\try.bas(12:016):  THEN
expected

tt = parse$(errmsg, any "()", 2)           ' Get the (nnn:nnn) value
errlin = val(tt)                          ' Line number preceeds the :
tt = mid$(tt, instr(tt, ":") + 1)         '
errcol = val(tt)                          ' Col number follows the :
errmsg = parse$(errmsg, "):", 2)          ' Message follows the ):

'----- Issue an error message and set the cursor to the error line
Set_Msg(fail, "Line: " + format$(errlin) + " Col: " + format$(errcol) + " " +
errmsg)
tt = Get_Line$(errlin + 1)                ' Get the error line
if errcol > len(tt) then                  ' Long enough to hi-lite?
    tt = lset$(tt, errcol)               ' Lengthen it
    Set_Line(errlin + 1, tt)             ' Stuff it back
end if
Set_Csr(errlin + 1, errcol, 1)            ' Hi-light the compiler's error
location
halt                                      ' Done
-----

```

```

' RM.MACRO
' Find rightmost occurrence of a string
' Syntax: RM string
'
' Author: George Deluca
'
dim hicol, hiline as number value 0

if Get_Arg$(0) = "" then halt(fail, "Missing search argument")

SPF_Cmd("FIND FIRST " & Get_Arg$(0))           ' Issue 1st FIND cmd
do while Get_RC = 0                             ' while found
    if Get_Find_Col > hicol then                 ' Save hi-water mark
        hicol = Get_Find_Col
        hiline = Get_Find_LPtr
    end if
    SPF_Cmd("RFIND")                           ' Look some more
loop

if hiline = 0 then                             ' Find anything?
    halt(fail, "String not found")              ' No, tell of error
else                                             ' Yes
    Set_Csr(hiline, hicol, Get_Find_Len)        ' Set cursor to it
    halt("Found", Get_Arg$(0), "rightmost in col:", format$(hicol)) ' Issue good
message
end if

```

```

-----
----

' FMBkDisp.macro
'----- Display Backup status for all files in the File Manager list
' Created: George Deluca - June 1, 2019
'
'----- Check environment
if isfalse Is_FM then halt(8, "FMBkDisp: Not running in File Manager")
if isfalse Is_Primary_Cmd then halt(8, "FMBkDisp: Not running as Primary Command")

dim i, j as long
dim t as string

'----- Display BACKUP status for each file
for i = 1 to FMGet_FCount when FMGet_FType(i) = FM_EQU_File ' Loop through Files
    j = FMGet_Backup_Versions(i)                             ' Get # backups
    t = lset$(FMGet_Abbrevname$(i, 20), 20)                  ' Get short fixed length
    filename
    t += " has " + iif$(j = 0, "no Backups", format$(j) + " Backup" + iif$(j > 1, "s",
    ""))
    FMSet_Msg(i, t)                                           ' Display the message
next i                                                         ' End of file loop
halt(0, "FMBkDisp: Display complete")

```

```

-----
----

' FMJoin.macro
'----- Join selected files in a File Manager session and open in a new Edit session
'----- If no macro operand, the collected data will be opened
'----- in a Clipboard session
'-----
'----- If an operand, it is assumed to be a filename and the
'----- collected data will be created in that file and the file
'----- opened in a normal Edit session.
'-----
'----- Select files with a Z line command (Could use any character)
'-----
' Created: George Deluca - June 3, 2019
'
USES "FILE"

```



```

'----- Check environment
if isfalse Is_FM then halt(8, "FMJoin: Not running in File Manager")
if isfalse Is_Primary_Cmd then halt(8, "FMJoin: Not running as Primary Command")

'----- Declare variables
dim FCount as long = FMGet_FCount
dim i, j as long
dim FName, FullName as string = "" ' Filename to null (CLIP mode)
dim CB, lText as string = "" ' Set strings to null
dim IFile, OFile as DWORD ' File handles

'----- See if an Output filename specified, if so, get it's name
if Get_Arg_Count > 0 then FName = Get_Arg$(1) ' An operand? Save as FName

'----- Scan for selected files in the list
for i = 1 to FCount when FMGet_FType(i) = FM_EQU_File ' Loop through Files

    if ucase$(trim$(FMGet_Cmd$(i))) = "Z" then ' A selected file?
        FullName = FMGet_Path$(i) + FMGet_FileName$(i) ' Build full filename

        '----- Open Output if not CLIP mode
        If FName <> "" and OFile = 0 then ' If O/P file and not opened
            if instr(FName, "\") = 0 then ' If FName unqualified
                FName = FMGet_Path$(i) + FName ' Add the Input path
            end if
            OFile = File_Open(FName, "OUTPUT") ' Open the output
            If OFile = 0 then Halt(8, "FMJoin: Can't open: " + FName)
        end if

        '----- Open the Input file
        IFile = File_Open(FullName, "INPUT") ' Open the file
        If IFile = 0 then Halt(8, "Can't open: " + FullName)

        '----- Read a single Input file
        Do while isfalse File_EOF(IFile) ' Read the data
            if FName = "" then ' If in CLIP mode
                CB += File_LineInput(IFile) + $CRLF ' Add to CB string
            else
                lText = File_LineInput(IFile) ' Read a line
                File_LinePrint(OFile, lText) ' Write to the output
            end if
        loop

        '----- Close Input, clear Select character
        File_Close(IFile)
        FMSet_Cmd(i, " ") ' Clear the selection char
        incr j ' Count files done
    end if
next i
if FName <> "" then File_Close(OFile) ' Close O/P file if we have one

'----- Did we accomplish anything?
if j = 0 then Halt(0, "FMJoin: No files were selected")

'----- Invoke via the Clipboard and CLIP
if FName = "" then ' In CLIP mode?
    ClipBoard_SetText(CB) ' Write to the Clipboard
    SPF_Post_Do("(Home) (EraseEOL) [CLIP] (Enter)") ' Invoke CLIP
else
    '----- Or invoke via a new edit session
    FName = SPF_Quote$(FName) ' Put FName in quotes
    SPF_Post_Do("(Home) (EraseEOL) [EDIT " + FName + "] (Enter)") ' Invoke EDIT
end if
Halt(0)

```


```
' FMBkAll.macro
'----- Backup all files in the File Manager list which have no existing backup
'-----
' Created: George Deluca - June 1, 2019
'
'----- Check environment
if isfalse Is_FM then halt(8, "FMBkAll: Not running in File Manager")
if isfalse Is_Primary_Cmd then halt(8, "FMBkAll: Not running as Primary Command")

dim i, j as long

'----- Scan file list for ones not backed up
for i = 1 to FMGet_FCount when FMGet_FType(i) = FM_EQU_File ' Loop through Files
    if FMGet_Backup_Versions(i) = 0 then ' If no current backups
        SPF_FM_LCmd(i, "BACKUP") ' Then request one
        incr j ' Count it
    end if ' End of no backups
next i ' End of file loop

'----- Issue appropriate message
halt(0, iif$(j = 0, "FMBkAll: All files had existing Backup(s)", "FMBkAll: " +
format$(j) + " Backups requested"))
```

Created with the Personal Edition of HelpNDoc: [Free help authoring environment](#)

thinBasic Essentials

Created with the Personal Edition of HelpNDoc: [Leave the tedious WinHelp HLP to CHM conversion process behind with HelpNDoc](#)

Overview

The macro language used by SPFLite is **thinBasic** (www.thinBasic.com) which is an interpreted variation of PowerBasic (www.PowerBasic.com). Both **thinBasic** and SPFLite itself are written in PowerBasic, which simplifies the linkage conventions between them. The interface between SPFLite and **thinBasic** is quite straightforward, which makes possible a very efficient implementation.

thinBasic is Copyright 2004 - 2013 by thinBasic

PowerBasic is Copyright by PowerBasic, Inc. 2061 Englewood Road Englewood, FL 34223

The SPFLite install will install the components of **thinBasic** needed to operate as the macro scripting language. It does **not** do a full install of **thinBasic**. If you plan to, or wish to explore **thinBasic** as a normal scripting language, you should obtain and install the full product from the **thinBasic** web site.

The Basic Language

The material following this point is meant as a quick summary of the Basic language used. The definitive answer as to syntax, usage etc. is the **thinBasic** Help document itself. You can reach the **thinBasic** Help file from within SPFLite by entering the command `HELP THINBASIC`

The **thinBasic** Help document is quite large and includes information on all aspects of **thinBasic**, including all the add-on modules, Software Developer Kit, etc. We have tried to provide the essentials for most coding tasks here to simplify things. Just don't consider it as the final word on what **thinBasic** has to offer.

We are mindful that **thinBasic** is a large language, one that offers everything you need to write macros. You may (rightly) conclude that it's "more than you need" or even "too much", but it's not likely you will say that it's "less than you need". The overview we provide here is presented with the aim of not overwhelming you, but giving you (mostly) only what you need to understand the principles and concepts to get some useful work done.

Because **thinBasic** is so large, we don't claim to have total knowledge or understanding of every last one of its inner workings. We may end up learning as much about **thinBasic** from you as you learn from us. This will be a learning experience for everyone, but it's going to be interesting !

Data Types

Although **thinBasic** supports most normal data types, unless you have some very particular requirement we suggest you use only the following three.

STRING

For the storage of normal text data. Strings are dynamically sized, and can contain strings (in theory) up to 2 GB characters long. In practice, string lengths are limited by available memory.

NUMBER

For the storage of numerical data. Number variables are Extended Precision floating point data of 80 bits in the range of 3.4×10^{-4932} to 1.2×10^{4932} . This format is more than capable of handling any of your numeric variable requirements. And as **thinBasic** uses this format internally for numeric calculations, it avoids repetitive internal data conversions if numeric variable are stored in this format.

LONG

For simple integer values, LONG is faster than the more generalized NUMBER.

Defining Variables

Every variable used in your macro must be declared before usage.

Variables are declared using the DIM keyword. The DIM keyword works just like standard Basic DIM except that a VALUE clause followed by a numeric or string expression can be entered. If the VALUE clause is used, and multiple variable names are specified, every variable will be initialized to the same value in the VALUE expression.

The VALUE expression can be any Basic expression, including function calls and the constants TRUE and FALSE>

Syntax

```
DIM VarName [ , Varname [ ,...]] AS NUMBER [ VALUE numeric-expression ]  
DIM VarName [ , Varname [ ,...]] AS STRING [ VALUE string-expression ]  
DIM AUTO
```

Examples

Code	Comment
DIM MyVar AS NUMBER	Define MyVar as a Number
DIM Var1, Var2 AS STRING VALUE "AAA"	Define Var1 and Var2 as STRINGs whose initial value is "AAA"
DIM Var2 AS LONG VALUE 100	This will declare an integer variable (Var2) whose initial value will be 100.

--	--

Operators

The macro language supports the following arithmetic and relational operators:

Arithmetic

+	Addition (for numeric expressions, concatenation for strings)
-	Subtraction
*	Multiplication
/	Division
\	Integer Division
^	Exponentiation
&	String concatenation (also see + above)

Relational

AND	And relationship
OR	Or relationship
NOT	Negation
<	Less than
<=	Less than or Equal
>	Greater than
>=	Greater than or Equal
<>	Not equal
=	Equal

Compound Numeric Operators

+=	Add variable value to right expression before assigning result back to variable
-=	Subtract variable value to right expression before assigning result back to variable
*=	Multiply variable value to right expression before assigning result back to variable
/=	Divide (floating point version) variable by right expression before assigning result back to variable
\=	Divide (integer version) variable by right expression before assigning result back to variable

Compound String Operators

- +=** Add variable value to right expression before assigning result back to variable
- &=** Add variable value to right expression before assigning result back to variable
- .=** Add variable value to right expression before assigning result back to variable

Implicit Assignment Operators

Implicit assignment takes place when a statement starts with a variable name followed by an operator.

For example:

B + 2

is considered as an implicit assignment to variable B and is equivalent the one of the below statement statements:

B = B + 2

B += 2

Implicit assignment is actually supported only for scalar numeric variables.

Supported operators are: **+** **-** ***** **/** ****

Created with the Personal Edition of HelpNDoc: [Easily create Help documents](#)

Labels

In some BASIC variants, you are permitted to define labels using a user-defined word, followed by a colon. These labels are referenced by GOTO and GOSUB statements.

thinBasic does not support GOTO or GOSUB statements, and you cannot define statement labels.

Created with the Personal Edition of HelpNDoc: [Effortlessly create a professional-quality documentation website with HelpNDoc](#)

Language Keywords

Created with the Personal Edition of HelpNDoc: [Maximize Your Documentation Capabilities with a Help Authoring Tool](#)

Flow Control

DO / LOOP / UNTIL	<pre>DO [{{WHILE UNTIL} Expression }} ... your code [EXIT DO] ... [ITERATE DO] ... your code ... LOOP [{{WHILE UNTIL} Expression }}</pre>
--------------------------	--

Expression is a numeric expression, in which non-zero values represent logical TRUE, and zero values represent logical FALSE.

If *Expression* contains string values, these must be part of a larger expression that results in a numeric value that can be treated as TRUE or FALSE. For example, the expression `A = "X"` produces a TRUE or FALSE value, but a simple string like "X" cannot be used as an expression in condition tests like WHILE or UNTIL. (That is, numbers can be converted to a "*truth value*", but strings cannot be.)

DO/LOOP statements are quite flexible. They allow you to create loops with the test for the terminating condition at the top of the loop, the bottom of the loop, both places, or none of the above.

The **WHILE** and **UNTIL** keywords are used to add tests to a **DO/LOOP**. Use the **WHILE** if the loop should be repeated if *expression* is **TRUE**, and terminated if *expression* is **FALSE**. **UNTIL** has the opposite effect; that is, the loop will be terminated if *expression* is **TRUE**, and repeated if **FALSE**.

When **WHILE** or **UNTIL** appears after the **DO** keyword, the test is performed **before** executing the body of the do-loop block. When **WHILE** or **UNTIL** appears after the **LOOP** keyword, the test is performed **after** executing the body of the do-loop block. It is possible to put a "before test" and an "after test" on the same do-loop block, though usually only one or the other is needed.

Note that **DO** is ended by **LOOP** and not by **END DO**.

An **EXIT DO** statement can be used to force an exit from a loop regardless of the **WHILE** / **UNTIL** condition. (**EXIT DO** is like a **break** or **leave** statement in some languages.)

An **ITERATE DO** statement transfers control to the bottom of the loop and triggers condition evaluation. (**ITERATE DO** is like a **continue** statement in some languages.) Because **ITERATE DO** creates the equivalent of a "backward-reference GOTO" statement, great care is needed to ensure you do not create a logic error that would cause the program to be "stuck in a loop".

FOR / NEXT	<pre>FOR Counter = Start TO Stop [STEP Increment] ... your code [EXIT FOR] ... [ITERATE FOR] ... your code ... NEXT [Counter]</pre>
-------------------	--

When a **FOR** statement is encountered, *start* is assigned to *Counter*, and *Counter* is tested to see if it is greater than (or, for negative *increment*, less than) *stop*. If the initial *Start* value makes the "stopping condition" immediately true, the body of the FOR/NEXT loop is not executed at all. If the initial *Start* value does not make the "stopping condition" immediately true, the statements within the **FOR/NEXT** loop are executed, *increment* is added to *Counter*, and *Counter* is again tested against *stop*. The statements in the loop are executed repeatedly until the stopping condition is true, at which time control passes to the statement immediately following the **NEXT**.

Counter is a numeric variable serving as the loop counter.

Start is a numeric expression specifying the value initially assigned to *Counter*.

Stop is a numeric expression giving the value that *Counter* must reach for the loop to be terminated.

Increment is an optional numeric expression defining the amount by which *Counter* is incremented with each loop execution. If not specified, *Increment* defaults to 1.

Note that **FOR** is ended by **NEXT** and not by **END FOR**.

An **EXIT FOR** statement can be used to force an exit from a loop regardless of stopping condition. (**EXIT FOR** is like a **break** or **leave** statement in some languages.)

An **ITERATE FOR** statement transfers control to the bottom of the loop and triggers evaluation of the stopping condition. (**ITERATE FOR** is like a **continue** statement in some languages.)

Because **ITERATE FOR** creates the equivalent of a "backward-reference GOTO" statement, great care is needed to ensure you do not create a logic error that would cause the program to be "stuck in a loop".

It is possible to modify the *Counter* variable inside the FOR/NEXT loop using assignment statements. However, experience has shown that such techniques often result in programming logic errors, and is thus not recommended. Instead of modifying the *Counter* you could do one of the following:

- Code an inner IF statement to selectively execute statements in a given loop iteration
- Use the EXIT FOR statement to leave the loop altogether
- Use the ITERATE FOR statement to begin the next iteration of the for-loop. Because of the potential risk of being "stuck in a loop" by using this statement, it is often safer to enclose a group of statements within your loop by an inner IF block, rather than trying to "skip" them using an ITERATE FOR statement.

NOTE The **NEXT** statement **does not** require the name of the *counter* variable. *Counter* is optional and is treated as comments.

IF / THEN / ELSE / ELSEIF / END IF	<pre>IF <i>Expression</i> THEN ... your code ... [ELSEIF <i>Expression</i> THEN ... your code ...] [ELSE ... your code ...] END IF</pre>
NOTE:	<p>Macro script does not support the single line IF / ELSE statement. The ELSE clause will be ignored.</p> <p>i.e. IF A = B then C = D else C = E will not function as you expect.</p>

In executing **IF** blocks, the truth of the *expression* in the initial **IF** statement is checked first. If it evaluates to **FALSE** (zero), each of the following **ELSEIF** statements is examined in order. There

can be as many **ELSEIF** statements as desired. As soon as one is found to be **TRUE** (non-zero), the statement(s) following the associated **THEN** and before the next **ELSEIF** or **ELSE** will be executed.

Execution then jumps to the statement just after the terminating **END IF** without making any further tests. If none of the test expressions evaluates to **TRUE**, the statement(s) in the **ELSE** clause (which is optional) are executed.

IF blocks must be terminated with a matching **END IF** statement. Note that the **END IF** statement requires a space and the **ELSEIF** statement does not.

SELECT / CASE / END SELECT	SELECT CASE expression CASE testlist {statements} [CASE testlist {statements}] [CASE ELSE {statements}] END SELECT
---	---

testlist is one or more tests, separated by commas, to be performed on expression. expression can be of type **STRING** or **NUMBER**.

When a **SELECT** statement is encountered, expression is evaluated using the testlist in the first **CASE** clause. If the evaluation is **FALSE**, the evaluation is repeated using the next testlist. As soon as an evaluation is **TRUE** (non-zero), the statements following that **CASE** clause are executed, up to the next **CASE** clause.

Execution then passes to the statement following the **END SELECT** statement. If none of the evaluations is **TRUE**, the statements following the optional **CASE ELSE** clause are executed.

The tests that may be performed by a **CASE** clause include: equality, inequality, greater than, less than, and range ("from-to") testing. The **SELECT CASE** block can do string or numeric tests, but these cannot be interchanged.

Examples of numeric CASE clause tests:

SELECT CASE numeric_expression

CASE > b	' relational; is expression >
b?	
CASE 14	' equality (= assumed); is
equal to 14?	
CASE b TO 99	' range; is it between
variable b and 99	
CASE 14, b	' two equality tests; equal
to 14 or b	

Examples of string CASE clause tests:

SELECT CASE string_expression

CASE > b\$	' relational; is expression >
b\$?	
CASE "X"	' equality (= is assumed); is
equal to "X"?	
CASE "A" TO "C"	' range; is between "A" and
"C"	
CASE "Y", b\$	' two equality tests; is

equal to "Y" or b\$?

WHILE / WEND	<pre>WHILE numeric_expression ... your code [EXIT WHILE] ... [ITERATE WHILE] ... your code ... WEND</pre>
---------------------	---

If `numeric_expression` is **TRUE** (it evaluates to a non-zero value), all of the statements between the **WHILE** and the terminating **WEND** are executed. Control then jumps back to the **WHILE** statement and repeats the test. If it is still **TRUE**, the enclosed statements are executed again. This process is repeated until the test expression evaluates to zero, or an **EXIT WHILE** statement is encountered. In either case, execution passes to the statement following **WEND**.

If `numeric_expression` evaluates to **FALSE** (zero) on the first pass, none of the statements in the loop are executed.

Loops built with **WHILE/WEND** statements can be nested (enclosed within each other). Each **WEND** matches the most recent unmatched **WHILE**.

Note that **WHILE** is ended by **WEND** and not by **END WHILE**.

Be aware that the **WHILE/WEND** statement is somewhat redundant as a language feature. You can do exactly the same thing (and more) using a **DO WHILE/LOOP**. As a coding style, you may wish to standardize on using **DO** statements rather than **WHILE** statements.

STOP HALT	<pre>STOP HALT([ret-code,] [str-msg] [,str- msg] ...)</pre>
--------------------	--

Stops execution of the script. In a stand-alone **thinBasic** program, **STOP** would halt the program execution. Because **thinBasic** is being used as a macro script engine, **STOP** returns control to SPFLite, which returns you to the edit session from which you originally launched the **thinBasic** macro.

The **HALT** command optionally allows you to set a macro Return code and message before terminating.

String Handling

Note: The following functions should not be considered a complete list of all available **thinBasic** string functions, but this set will probably do for 99% of your requirements. If time permits, explore the full **thinBasic** Help document to review the complete set. Enter `HELP thinBasic` at the SPFLite command prompt to open the **thinBasic** help file.

ASC	<code>Number = ASC(string-expression [,position])</code>
------------	--

ASC returns the character code of a particular character in the *string-expression*. The returned value will be in the range of 0 to 255.

The optional *position* parameter determines which character is to be checked. The first character is one, the second two, etc. If the *position* parameter is missing, the first character is presumed. If *position* is negative, ASC counts from the end of the string in reverse. That is, -1 specifies the last character, -2 specifies the second to last character, etc.

CHOOSE\$	<code>String = CHOOSE\$(index, string-expression [,string-expression] ...)</code>
-----------------	--

Return one of several values, based upon the value of an index. It returns one of the parameters based upon the value of *index*. That is, if *index* is one, choice1 is returned. If two, choice2 is returned, etc. If *index* is not equal to one of the choice values, "" (zero-length string) is returned.

CHR\$	<code>String = CHR\$(expression [,expression] [,...]) String = CHR\$(numvar1 to numvar2 [,...])</code>
--------------	--

This function creates a string of characters. Arguments must be normal characters, or codes in the range of 0 to 255.

CHR\$ creates and returns a string. There are two forms of arguments available, and they may be intermixed in a single **CHR\$** function. The created string may contain no characters, one character, or multiple characters, depending upon the arguments you use. You may specify any number of arguments for this function.

If the argument is a numeric expression, it is translated into the character defined by that number. A character code of -1 is treated as a special case. If you use it as an argument, **CHR\$** returns an empty (zero length) string for that character. For example, **CHR\$(65, -1, 66)** returns "AB".

CHR\$(numvar1 TO numvar2)
returns a sequence of all characters
from **CHR\$(numvar1)** through
CHR\$(numvar2) inclusive. The
characters may be ascending or
descending in sequence. For example,
CHR\$(65 TO 70) returns the

string "ABCDEF". **CHR\$ (52 TO 50)** returns the string "432", and **CHR\$ (65 TO 65)** returns the string "A".

CHR\$ complements the **ASC** function, which returns the numeric character code of a nominated character in a string.

\$CRLF	\$CRLF
---------------	--------

This function returns a Carriage Return / Line Feed pair. Equivalent to **CHR\$ (13, 10)**

\$DQ	\$DQ
-------------	------

This function returns a Double-Quote character Equivalent to **CHR\$ (34)**

CSET\$	Result-String = CSET\$(string-expression, str-len [USING string-expression2])
---------------	---

Return a string containing a centered (padded) string. **CSET\$** centers the string *string-expression* into a string of *str-len* characters.

If *string-expression2* is "" (zero-length string) or is not specified, **CSET\$** pads string-expression with space characters. Otherwise, **CSET\$** pads the string with the first character of string-expression2.

If *string-expression* is shorter than *str-len*, **CSET\$** centers *string-expression* within *Result-String*, padding both sides as described above; otherwise, **CSET\$** returns the left-most *str-len* bytes of *string-expression*.

DATE\$	Result-String = DATE\$
---------------	------------------------

Returns the system date. You can assign **DATE\$** to a string variable, which stores 10 characters in the form "mm-dd-yyyy", where mm represents the month, dd the day, and yyyy the year.

EXTRACT\$	Result-String = EXTRACT\$([start,] string-expression, [ANY] Match-Str
------------------	---

EXTRACT\$ returns a sub-string of *String-Expression*, starting with its first character (or the character specified by *start*) and up to (but not including) the first occurrence of *Match-Str*.

If *Match-Str* is not present in *String-Expression*, or either string parameter is nul, all of *String-Expression* is returned.

start is the optional starting position to begin extracting. If *start* is not specified, it will start at

position 1. If *start* is zero, or beyond the length of *String-Expression*, a nul string is returned. If *start* is negative, the starting position is counted from right to left: if -1, the search begins at the last character; if -2, the second to last, and so forth.

String-Expression is the string from which to extract. *Match-Str* is the string expression to extract up to. **EXTRACT\$** is case-sensitive.

If the **ANY** keyword is included, *Match-Str* specifies a list of single characters to be searched for individually, a match on any one of which will cause the extract operation to be performed up to that character.

A similar function to **EXTRACT\$** is **PARSE\$**, which extracts delimited substrings from a string.

FORMAT\$	Result-String = FORMAT\$(numeric-expression [, Format-Mask])
-----------------	---

Returns a string version of *numeric expression*, formatted according to the *Format-Mask* operand.

If no *Format-Mask* operand is present, the number will be returned as a maximum of 16 significant digits.

Note: If you wish to just concatenate a string and a numeric value without formatting, this can be done with a simple expression of the form *string-constant* + *numeric-expression* or *string-constant* & *numeric-expression*.

Format-Mask

Format characters that will determine how *numeric-expression* should be formatted. This expression is termed the mask. There may be up to 18 digit-formatting digits on either side of the decimal point. The mask may not contain literal characters unless each character is preceded with a backslash (\) escape character, or the literal characters are enclosed in quotes.

Format-Mask may contain one, two or three formatting masks, separated by semicolon (;) characters:

One mask	If <i>Format-Mask</i> contains just one format mask, the mask is used to format all possible values of <i>numeric-expression</i> . For example: x\$ = FORMAT\$(z, "000.00")
Two masks	If <i>Format-Mask</i> contains two format masks, the first mask is used for positive values (≥ 0), and the second mask is used for negative values (< 0). For example: x\$ = FORMAT\$(-100, "+00000.00;-000")
Three masks	If <i>Format-Mask</i> contains three masks, the first mask is used for positive values (> 0), the second mask for negative values (< 0), and the third mask is used if <i>numeric-expression</i> is zero (0). For example: FOR y = -0.5 TO 0.5 STEP 0.5 x\$ = FORMAT\$(y, "+.0;-.0; .0") NEXT y

Digit placeholders in a mask do not have to be contiguous. This allows you to format a single

number into multiple displayed parts. For example:

```
A$ = FORMAT$(123456, "00\:00\:00") ' 12:34:56
```

The following table shows the characters you can use to create the user-defined format strings (masks) and the definition of each formatting character:

Character	Definition
Empty string	["" (a zero-length string)] No formatting takes place. The number is formatted similarly to space that STR\$ applies to non-negative numbers. <div> <div>A\$=FORMAT\$(0.2) ' .200000002980232</div> <div>A\$=FORMAT\$(0.2!, "") ' .200000002980232</div> <div>A\$=FORMAT\$(123) ' 123</div> <div>A\$=FORMAT\$(123456) ' 123456</div> </div>
0	<div>[zero] Digit placeholder. A digit or 0 will be inserted in that position.</div> <div> <p>If there is a digit in numeric-expression in the position where the 0 appears in the format s Otherwise, return "0". If the number being formatted has fewer digits than there are zeros (point) in the format expression, leading or trailing zeros are added. If the number has more decimal point than there are zeros to the right of the decimal point in the format expression as many decimal places as there are zeros in the mask.</p> <p>If the number has more digits to the left of the decimal point than there are zeros to the left format expression, the extra digits are displayed without truncation. If the numeric value is symbol will be treated as a decimal digit. Therefore, care should be exercised when displaying this placeholder style. In such cases, it is recommended that multiple masks be used.</p> <p>' Numeric padded with leading zero characters</p> <div>A\$ = FORMAT\$(999, "00000000") ' 00000999</div> </div>
#	<div>[Number symbol] Digit placeholder. If there is a digit for this position, it is replaced with a specified character.</div> <div> <p>Unlike the 0 digit placeholder, if the numeric value has the fewer digits than there are # character decimal placeholder, then it will either:</p> <p>a) Omit this character position from the final formatted string; or</p> <p>Substitute a user-specified replacement character if one has been defined (see the asterisk information). To specify leading spaces, prefix the mask with "*" (asterisk and a space character).</p> <p>For example:</p> <p>' No leading spaces and trailing spaces</p> <div>A\$ = FORMAT\$(0.75, "####.####") ' 0.75</div> <p>' Up to 3 Leading spaces before decimal</p> <div>A\$ = FORMAT\$(0.75, "* ##.####") ' 0.75</div> <p>' Using asterisks for padding characters</p> </div>

A\$ = FORMAT\$(0.75, "*=##.###") ' ==0.75=

FORMAT\$ may also return a string that is larger than the number of characters in the mask.

A\$ = FORMAT\$(999999.9, "#.#") ' 999999.9

.

[period] Decimal placeholder. Determines the position of the decimal point in the resultant string.

If any numeric field is specified to the left of the decimal point, at least one digit will always appear to the left of the decimal point. The zero is not considered to be a "leading" zero if it is the only digit to the left of the decimal point. The period character in the *Format-Mask* string will produce undefined results.

%

[percent] Percentage placeholder. *numeric-expression* will be multiplied by 100, and adds a percent sign. For example:

x\$ = FORMAT\$(1 / 5, "0.0%") ' 20.0%

,

[comma] Thousand separator. Used to separate thousands from hundreds within a number. The comma must appear to the left of the decimal point. In order to be recognized as a format character, the comma must appear after a digit placeholder character (also see Restrictions below).

A\$ = FORMAT\$(1234567, "#,") ' 1,234,567

A\$ = FORMAT\$(12345, "#,.00") ' 12,345.00

A\$ = FORMAT\$(12345, "#.00,") ' 12,345.00

A\$ = FORMAT\$(1212.46, "\$00,000.00") ' \$01,212.46

A\$ = FORMAT\$(1000, """"#"#,"") ' #1,000

A\$ = FORMAT\$(1234567, "0,") ' 1,234,567

*x

[asterisk] Digit placeholder and fill-character. Instructs FORMAT\$ to insert a digit or character if there is a digit in *numeric-expression* at the position where the * appears in the format string; otherwise, the "x" character is used (where "x" represents your own choice of character). The asterisk (*) is a digit placeholder (#) fields.

A\$ = FORMAT\$(9999.9, "\$**#####,.00") ' \$**9,999.90

A\$ = FORMAT\$(0, "\$*=###0,.00#") ' \$====0.00=

A\$ = FORMAT\$(0, "\$* #####0,.00") ' \$ 0.00

E- e- E+
e+

[e] Scientific format. FORMAT\$ will use scientific notation in the formatted output. Use E- or e- in front of negative exponents. Use E+ or e+ to place a minus sign in front of negative exponents. Use E- or e- in front of non-negative exponents.

In order to be recognized as a format sequence, the E-, e-, E+, or e+ must be placed between two characters. For example:

A\$ = FORMAT\$(99.999, "0.0E-##") ' 1.0E2

A\$ = FORMAT\$(-99.999, "0.0E-##") ' -1.0E2

A\$ = FORMAT\$(99.999, "0.0E+##") ' 1.0E+2

A\$ = FORMAT\$(-99.999, "0.0E+##") ' -1.0E+2

A\$ = FORMAT\$(0.1, "0.0e+##") ' 1.0e-1

" [double-quote] Quoted string. **FORMAT\$** treats all characters up to the next quotation mark as digit placeholders or format characters. Also see backslash. For example:

```
A$ = FORMAT$(5.55, ""XYZ=""#.##\#" ) ' XYZ=5.55#
```

```
A$ = FORMAT$(25, ""x=""#" ) ' x=25
```

```
A$ = FORMAT$(999, ""Total ""#" ) ' Total 999
```

\x [backslash] Escaped character prefix. **FORMAT\$C** treats the character "x" immediately following the backslash as a literal character rather than a digit placeholder or a formatting character. Many characters have special meaning and cannot be used as literal characters unless they are preceded by a backslash.

The backslash itself is not copied. To display a backslash, use two backslashes (\\). To display a double-quote, use two double-quote characters.

IIF\$	Result-String = IIF\$(numeric-expression, true-string, false-string)
--------------	---

IIF\$ returns one of two values based on *numeric-expression*. If *numeric-expression* evaluates to TRUE (non-zero), the *true-string* is returned, else the *false-string* is returned. Compare to the **IIF** function.

INSTR	Result-var = INSTR([Position,] Main-Str, [ANY] Match-Str)
--------------	---

INSTR returns the position of *Match-Str* within *Main-Str*. The return value is indexed to one, while zero means "not found".

Position specifies the character position to begin the search. If *Position* is one or greater, *Main-Str* is searched left to right. The value one starts at the first character, two the second, etc. If *Position* is -1 or less, *Main-Str* is searched from right to left. The value -1 starts at the last character, -2 the second to last, etc. If *Position* is not given, the default value of +1 is assumed.

```
x& = INSTR("xyz", "y") ' returns 2
```

```
x& = INSTR("xyz", "a") ' returns 0
```

```
a$ = "My Dog" : b$ = " "
```

```
x& = INSTR(a$, b$) ' returns 3
```

It is important to note that in all cases, even when *Position* is negative, the return value of **INSTR()** is the absolute position of the match, from left to right, starting with the first character.

ANY

If the **ANY** keyword is included, *Match-Str* specifies a list of single characters. **INSTR** searches for each of these characters individually. As soon as any one of these characters is found, **INSTR** returns the position of the match.

```
x& = INSTR(-2, "efcdef",  
ANY "ef") returns a  
result of 5
```

INSTR is case-sensitive, meaning that upper-case and lower-case letters must match exactly in *Match-Str* and *Main-Str*.

LCASE\$	Result-str = LCASE\$(string-expression)
----------------	---

Returns a lowercase version of the specified *string-expression*.

LEFT\$	Result-str = LEFT\$(string-expression, Num-chars)
---------------	--

Num-chars specifies the number of characters in *string-expression* to be returned.

LEFT\$ returns a string consisting of the left most *Num-chars* characters of its string argument. If *Num-chars* is greater than or equal to the length of *string-expression*, all of *string-expression* is returned. If *Num-chars* is zero, **LEFT\$** returns an empty string.

LEN	Result-var = LEN(string-expression)
------------	-------------------------------------

LEN will return the current length of *string-expression*.

LSET\$	Result-str = LSET\$(string-expression, str-len [USING pad-string])
---------------	--

LSET\$ left-aligns the string *string-expression* into a string of *str-len* characters.

USING

If *pad-string* is "" (a zero-length string) or is not specified, **LSET\$** pads *string-expression* with space characters. Otherwise, **LSET\$** pads the string with the first character of *pad-string*.

If *string-expression* is shorter than *str-len*, **LSET\$** left-justifies *string-expression* within *result-str*, padding the right side as described above; otherwise, **LSET\$** returns the left-most *str-len* bytes of *string-expression*.

LTRIM\$	Result-str = LTRIM\$(string-to-trim [, [ANY] Chars-to-trim])
----------------	---

String-to-trim is the string-expression from which to remove characters, and *Chars-to-trim* is the string-expression containing the characters to remove.

If *Chars-to-trim* is not specified, **LTRIM\$** removes leading spaces. **LTRIM\$** returns a substring of *String-to-trim*, from the first non-*Chars-to-trim* (or non-space) to the end of the string. If *Chars-to-trim* (or a space) is not present at the beginning of *String-to-trim*, all of *String-to-trim* is returned.

If the **ANY** keyword is included, *Chars-to-trim* specifies a list of single characters to be searched for individually. A match on any one of these as a leading character will cause the character to be removed from the result.

LTRIM\$ is case sensitive

MCASE\$	Result-str = MCASE\$(string-expression)
----------------	---

MCASE\$ returns a string equivalent to *string-expression*, except that the first letter of each word is capitalized, while the remaining characters are forced to lowercase. A word is considered to be a consecutive series of letters.

MID\$	Result-str = MID\$(string-expression, start-loc [, length])
--------------	---

The **MID\$** function returns a part of a string-expression. The *start-loc* specifies the location of the first character of the extracted string. The *length* specifies how many characters, starting at *start-loc* are to be returned.

If *length* is omitted, or there aren't enough characters in *String-expression*, all remaining characters are returned. If there are no characters at the *Start-loc* position, an empty string is returned.

PARSE\$	Result-str = PARSE\$(string-expression, [ANY] str-delimiter, index)
----------------	---

Returns a delimited field from *string-expression*. *string-expression* is the string to parse. If *string-expression* is "" (a zero-length string) or contains no delimiter character(s), the string is considered to contain exactly one field. In this case, **PARSE\$** will return *string-expression*.

The *str-delimiter* contains delimiter character(s). A delimiter is a character, list of characters, or string, that is used to mark the end of a field in string-expression. For example, if you consider a sentence to be a list of words, the delimiter between the words is a space (or punctuation). A delimiter is not considered part of a field, but as the divider between fields, so the delimiter is never returned by **PARSE\$**.

If *str-delim* is not specified or is "" (a zero-length string), standard comma-delimited (optionally quoted) fields are presumed. In this case only, the following parsing rules apply. If a standard field is enclosed in optional quotes, they are removed. If any characters appear between a quoted field and the next comma delimiter, they are discarded. If no leading quote is found, any leading or trailing blank spaces are trimmed before the field is returned.

Delimiters are case-sensitive, so capitalization may be a consideration.

ANY

If the **ANY** keyword is used, *str-delimiter* contains a set of characters, any of which may act as a delimiter character. If the **ANY** keyword is omitted, the entire *str-delimiter* string acts as a single delimiter.

index

A variable or expression that specifies the delimited field number to return. The first field is 1, and so on up to the maximum number of fields contained in string-expression, which may be determined with the **PARSECOUNT** function. If *index* is negative, string-expression is parsed from right to left. In this case, *index* = -1 returns the last field in string-expression, -2 returns the second to last, etc. If *index* evaluates to zero, or is outside of the actual field count, an empty string is returned.

PARSECOUNT	Result-str = PARSE\$(string-expression, [ANY] str-delimiter)
-------------------	--

Return the count of delimited strings in a *string-expression*. **PARSECOUNT** uses the same

rules as `PARSE$` in the determination of fields within string-expression. Individual fields within string-expression are evaluated, and the tally of the fields forms the result value.

It is important to note that `PARSECOUNT` may only be used with string data which contains variable length sub-fields, each of which is separated by a delimiter. To determine the count of fixed length data, divide the String-Expression length by the sub-field length. If this function is used with fixed length data, the results are undefined.

string-expression

This is the string to examine and parse. If String-Expression is "" (a zero-length string) or contains no delimiter character(s), the string is considered to contain exactly one sub-field. In this case, `PARSECOUNT` returns the value 1.

str-delimiter

This defines one or more characters to use as a delimiter. To be valid, the entire delimiter must match exactly, but the delimiter itself is never returned as part of the field.

If *str-delimiter* is not specified, or contains an empty string, special rules apply. The delimiter is assumed to be a comma. Fields may optionally be enclosed in quotes, and are ignored before the result string is returned. Any characters that appear between a quote mark and the next comma delimiter character are discarded. If no leading quote is found, any leading or trailing quotes are trimmed before the result string is returned.

ANY

If the **ANY** keyword is used, *str-delimiter* contains a set of characters, any of which may act as a delimiter character. If the **ANY** keyword is omitted, the entire *str-delimiter* string acts as a single delimiter.

REMOVE\$	Result-str = REMOVE\$(string-expression, [ANY] chars-to-remove)
-----------------	--

Return a copy of a string with characters or strings removed.

string-expression is the string from which to remove characters. *chars-to-remove* is the string expression to remove all occurrences of. If *chars-to-remove* is not present in *string-expression*, all of *string-expression* is returned intact.

ANY

If the **ANY** keyword is included, *chars-to-remove* specifies a list of single characters to be searched for individually, a match on any one of which will cause that character to be removed from the result.

REMOVE\$ is case-sensitive.

REPEAT\$	Result-str = REPEAT\$(number-of-times, string-expression)
-----------------	--

Returns a string consisting of multiple copies of the specified string. *number-of-times* is an expression, constant or variable, specifying the number of copies of *string-expression* to be included in the result.

string-expression is the string to be duplicated.

REPLACE\$	Result-str = REPLACE\$(string-
------------------	--------------------------------

	<code>expression, [ANY] match-string, new-string)</code>
--	---

Within a specified string , replace all occurrences of one string with another string. The **REPLACE\$** statement replaces all occurrences of *Match-String* in *string-expression* with *New-String*. The replacement can cause string-expression to grow or condense in size. *string-expression* must be a string variable; *Match-String* and *New-String* may be string expressions. **REPLACE\$** is case-sensitive. When a match is found, the scan for the next match begins at the position immediately following the prior match.

ANY

If you use the **ANY** option, within *String-expression*, each occurrence of each character in *Match-String* will be replaced with the corresponding character in *New-String*. In this case, *Match-string* and *New-String* must be the same length, because there is a one-to-one correspondence between their characters.

RIGHT\$	<code>Result-str = RIGHT\$(string-expression, Num-chars)</code>
----------------	---

Num-chars specifies the number of characters in *string-expression* to be returned.

RIGHT\$ returns a string consisting of the right most *Num-chars* characters of its string argument. If *Num-chars* is greater than or equal to the length of *string-expression*, all of *string-expression* is returned. If *Num-chars* is zero, **RIGHT\$** returns an empty string.

RSET\$	<code>Result-str = RSET\$(string-expression, str-len [USING pad-string])</code>
---------------	--

RSET\$ left-aligns the string *string-expression* into a string of *str-len* characters.

USING

If *pad-string* is "" (a zero-length string) or is not specified, **RSET\$** pads *string-expression* with space characters. Otherwise, **RSET\$** pads the string with the first character of *pad-string*.

If *string-expression* is shorter than *str-len*, **RSET\$** right-justifies *string-expression* within *result-str*, padding the left side as described above; otherwise, **RSET\$** returns the right-most *str-len* bytes of *string-expression*.

RTRIM\$	<code>Result-str = RTRIM\$(string-to-trim [, [ANY] Chars-to-trim])</code>
----------------	---

String-to-trim is the string-expression from which to remove characters, and *Chars-to-trim* is the string-expression containing the characters to remove.

String-to-trim is the *string-expression* from which to remove characters, and *Chars-to-trim* is the string expression specifying the characters that should be removed from the right hand side of *String-to-trim*.

If *Chars-to-trim* is not specified, **RTRIM\$** removes trailing spaces. **RTRIM\$** returns a substring of *String-to-trim*, from the beginning of the string to the character preceding the consecutive occurrences of *Chars-to-trim* (or space), which continues to the end of the

original string . If *Chars-to-trim* (or a space) is not present at the end of *String-to-trim*, all of *String-to-trim* is returned.

If the **ANY** keyword is included, *Chars-to-trim* specifies a list of single characters to be searched for individually - a match on any one of which as a trailing character will cause the character to be removed from the result.

RTRIM\$ is case-sensitive.

STR\$	Result-str = STR\$(num-expression [, digits])
--------------	--

Return the representation of a number in printable string form.

STR\$ returns the string form of a variable or expression in printable text form. *digits* is an optional expression specifying the maximum total number of digits to appear in the result. If *num-expression* is greater than or equal to zero, **STR\$** adds a leading space character; if *num-expression* is less than zero, **STR\$** adds a leading negation (minus) character.

For example, **STR\$(14)** returns a three-character string, of which the first character is a space, and the second and third are "1" and "4".

digits specifies the maximum number of significant digits (1 to 18) desired in the result.

The complementary function is **VAL**, which takes a string argument and returns the numeric equivalent.

Thus, **number = VAL(STR\$(number))**.

A numeric value may also be converted to a string with the **TSTR\$** function which returns the string without the leading blank or - character, or the **FORMAT\$** function. **FORMAT\$** is capable of many additional formatting options.

STRCONST\$	Result-str = STRCONST\$(str-expression)
-------------------	---

Return a string representing the mnemonic string passed as *str-expression*. *str-expression* may be any of "CRLF", "TAB", "CR", "LF", "FF", "VT", "SPC", "DQ", "NUL", "ESC"

STRDELETE\$	Result-str = STRDELETE\$(str-expression, start, count)
--------------------	--

Delete a specified number of characters from *str-expression*.

Returns a string based on copying *str-expression*, but with *count* characters deleted starting at position *start*. The first character in the string is position 1, etc.

STRINSERT\$	Result-str = STRINSERT\$(str-expression, ins-expression, position)
--------------------	--

Insert *ins-expression* string at a specified position within *str-expression*.

Returns a string consisting of the string expression *str-expression*, with the string expression *ins-expression* inserted at *position*. If *position* is greater than the length of *str-expression*, *ins-expression* is appended to *str-expression*. The first character in the string is position 1, etc.

STRREVERSE\$	Result-str = STRREVERSE\$(str-expression)
---------------------	---

Reverse the contents of *str-expression*.

TALLY	Num-var = TALLY(<i>str-expression</i> , [ANY] <i>Match-string</i>)
--------------	---

Count the number of occurrences of specified characters or strings within *str-expression* .

str-expression is the string in which to count characters. *Match-String* is the string expression to count all occurrences of. If *Match-String* is not present in *str-expression*, zero is returned. When a match is found, the scan for the next match begins at the position immediately following the prior match.

ANY

If the **ANY** keyword is included, *Match-String* specifies a list of single characters to be searched for individually: a match on any one of which will cause the count to be incremented for each occurrence of that character. Note that repeated characters in *Match-String* will not increase the tally. For example:

X = TALLY("ABCD" , ANY "BDB") ' returns 2, not 3

TALLY is case-sensitive, so be wary of capitalization.

TIME\$	Result-str = TIME\$
---------------	---------------------

Return a string with system time in the format of HH:MM:SS using 24-hour time.

TRIM	Result-str = TRIM\$(<i>str-expression</i> [, [ANY] <i>Chars-To-Trim</i>)
-------------	---

Removes leading and trailing characters or substrings from *str-expression*.

TRIM\$ combines the functionality of **LTRIM\$** and **RTRIM\$** into a single function. *str-expression* is the string expression from which to remove characters, and *Chars-To-Trim* is the string expression to remove leading and trailing occurrences. If *Chars-To-Trim* is not specified, **TRIM\$** removes leading and trailing spaces.

ANY

If the **ANY** keyword is included, *Chars-To-Trim* specifies a list of single characters to be searched for individually, a match on any one of which as a leading or trailing character will cause the character to be removed from the result.

TSTR\$	Result-str = TSTR\$(<i>num-expression</i> [, <i>digits</i>])
---------------	--

Return the representation of a number in printable string form.

TSTR\$ returns the string form of a variable or expression in printable text form. *digits* is an optional expression specifying the maximum total number of digits to appear in the result.

For example, **TSTR\$(14)** returns a two-character string, "14".

digits specifies the maximum number of significant digits (1 to 18) desired in the result.

The complementary function is **VAL**, which takes a string argument and returns the numeric

equivalent.

Thus, **number** = **VAL**(**TSTR\$**(**number**)).

A numeric value may also be converted to a string with the **FORMAT\$** function. **FORMAT\$** can optionally perform many additional formatting functions.

UCASE\$	Result-str = UCASE\$(string-expression)
----------------	---

Returns an uppercase version of the specified *string-expression*.

VAL	Num-var = VAL(string-expression)
------------	----------------------------------

Convert a text string to a numeric value.

The **VAL** function converts a string argument to a number. Leading white-space characters (spaces, tabs, carriage-returns, and linefeeds) are skipped and ignored. Evaluation of the number continues until a non-numeric character is found, or the end of the string is reached. If no number is found, the **VAL**() function returns zero (0). Format characters (like commas) are not allowed, and will cause early termination of the evaluation.

VAL interprets the letters "e" and "d" (and "E" and "D") as the symbols for exponentiation and scientific notation:

```
i& = VAL("10.101e3")      '
10101 ~ 10.101*(10^3)
```

```
j& = VAL("2D4")           '
20000 ~ 2 * (10 ^ 4)
```

Hexadecimal, Binary and Octal conversions

VAL can also be used to convert string arguments that are in the form of Hexadecimal, Binary and Octal numbers. Hexadecimal values should be prefixed with "&H" and Binary with "&B". Octal values may be prefixed "&O", "&Q" or just "&". If the string-expression contains a leading zero, the result is returned as an unsigned value; otherwise, a signed value is returned. For example:

```
i& = VAL("&HF5F3")        '
Hex, returns -2573 (signed)
```

```
j& = VAL("&H0F5F3")        '
Hex, returns 62963 (unsigned)
```

```
x& = VAL("&B0100101101")   '
Binary, returns 301 (unsigned)
```

```
y& = VAL("&O4574514")      '
Octal, returns 1243468 (signed)
```

Valid hex characters include 0 to 9, A to F (and a to f). Valid Octal characters include 0 to 7, and binary 0 to 1.

VAL stops analyzing *string-expression* when non-numeric characters are encountered. When dealing with Hexadecimal, Binary, and Octal number systems, the period character is

classified as non-numeric.

VERIFY	Num-var = VERIFY([start,] string-expression, Match-String)
---------------	---

Determine whether each character of a string is present in another string.

VERIFY returns zero if each character in *string-expression* is present in *Match-String*. If not, it returns the position of the first non-matching character in *string-expression*.

This function is useful for determining if a string contains only digits. For example, VERIFY(NUM, "0123456789") = 0 if NUM is a numeric string.

VERIFY is case-sensitive, so capitalization matters.

If *start* evaluates to a position outside of the string on either side, or if *start* is zero, **VERIFY** returns zero.

Numeric Handling

Note: The following functions should not be considered a complete list of all available **thinBasic** numeric functions, but this set will probably do for 99% of your requirements. If time permits, explore the full **thinBasic** Help document to review the complete set. Enter `HELP thinBasic` at the SPFLite command prompt to open the **thinBasic** help file.

ABS	Num-var = ABS(numeric-expression)
------------	-----------------------------------

Return the absolute value of a expression.

The absolute value of a number is its non-negative value . For example, the absolute value of -3 is 3, and the absolute value of +3 is also 3. The absolute value of 0 is 0.

ATN	Num-var = ATN(numeric-expression)
------------	-----------------------------------

Return the arctangent of an argument.

ATN returns the arctangent (Inverse Tangent) of *numeric-expression*; that is, the angle whose tangent is *numeric-expression*.

The result is in radians rather than degrees. To convert radians to degrees, multiply the radian value by 180/pi, or 57.29577951308232.

CEIL	Num-var = CEIL(numeric-expression)
-------------	------------------------------------

Convert a variable or expression into an value, by returning the smallest integral value that is greater than or equal to its argument.

The **CEIL** function rounds upward, returning the smallest integral value that is greater than or equal to *numeric-expression*.

For example, `y = CEIL(1.5)` places the value 2 into y.

CHOOSE	Num-var = CHOOSE(index, string-expression [,string-expression] ...)
---------------	--

Return one of several values, based upon the value of an index. It returns one of the parameters based upon the value of *index*. That is, if *index* is one, choice1 is returned. If two, choice2 is returned, etc. If *index* is not equal to one of the choice values, 0 is returned.

COS	Num-var = COS(numeric-expression)
------------	-----------------------------------

Return the cosine of an argument.

numeric-expression is an angle specified in radians. To convert radians to degrees, multiply by 57.29577951308232##. To convert degrees to radians, multiply by 0.0174532925199433.

COS returns a value that always ranges between -1 and +1 inclusive.

DECR	DECR numeric-variable
-------------	-----------------------

Decrement a variable by 1.

Note: A variable may also be decremented by the notation **variable -= 1**

EXP	Num-var = EXP(numeric-expression)
EXP2	
EXP10	

Return a base number raised to a power. The base is **e** for **EXP**, 2 for **EXP2**, and 10 for **EXP10**.

EXP returns e to the nth power, where n is a numeric variable or expression and e is the base for natural logarithms, approximately 2.718282. Among other uses, this provides a simple way to obtain the value of e itself:

e = EXP(1)

EXP2 (n) returns 2 to the nth power, where **n** is a numeric variable or expression.

EXP10 (n) returns 10 to the nth power, where **n** is a numeric variable or expression.

The **EXP** functions provide a convenient alternative to the ^ operator, which works with any base.

FIX	Num-var = FIX(numeric-expression)
------------	-----------------------------------

Truncate a number to an integer value.

FIX strips off the fractional part of its argument, and returns the integer part. Unlike **INT**, **FIX** does not perform any form of rounding or scaling.

FALSE	Same as literal value of 0
TRUE	Same as literal value of -1

FALSE and TRUE may be used anywhere the literal values of 0 and -1 can be used, including the VALUE clause of DIM statements.

Be aware that, while the value of TRUE is -1, a "true" value for purposes of conditional expressions such as in IF statements is a **non-zero** value, not just the value -1. This is similar to how conditions are tested in the C language. You can use TRUE and FALSE to set a flag variable, and you can test a flag for FALSE, but it's best not to test a flag for TRUE. Instead of using **IF flag = TRUE THEN**, you should simply say **IF flag THEN**. If you want to **force** a numeric expression to be exactly equal to either TRUE or FALSE, you can use the **thinBasic** function **IsTrue**.

FRAC	Num-var = FRAC(Numeric-expression)
-------------	------------------------------------

Return the fractional part of a number.

FRAC returns the number after the decimal point of a number or expression. **FRAC** rounds the result to fit the precision of Numeric Expression.

IIF	Num-var = IIF(Num-expression, true-part, false-part)
------------	--

Return one of two values based upon a TRUE/FALSE evaluation.

If *num-expression* evaluates to **TRUE** (non-zero), the *true-part* is returned, else the *false-part* is returned. *num-expression* is evaluated as a normal Boolean expression.

Both *true-part* and *false-part* should generate a normal numeric return value.
Compare to the **IIF\$** function.

INCR	INCR numeric-variable
-------------	-----------------------

Increment a variable by 1.

Note: A variable may also be incremented by the notation **variable += 1**

INT	Num-var = INT(Num-expression)
------------	-------------------------------

INT rounds *num-expression* to the largest integral value that is less than or equal to *num_expression*.

ISFALSE	ISFALSE(expression)
ISTRUE	ISTRUE(expression)

ISTRUE returns -1 (**TRUE**) when *expression* evaluates as non-zero; otherwise, it returns zero (**FALSE**). **ISFALSE** returns -1 when *expression* evaluates as 0 (**FALSE**); otherwise, it returns zero.

Truth table

operator	<i>expression</i>	Result
ISTRUE	= 0	0 (False)
ISTRUE	<> 0	-1 (True)
ISFALSE	= 0	-1 (True)
ISFALSE	<> 0	0 (False)

MAX	Num-var = MAX(Number1, [,Number2] ...)
------------	--

Return the argument with the largest (maximum) value.

This function takes any number of arguments and return the argument with the largest (maximum) value.

MIN	Num-var = MIN(Number1, [,Number2] ...)
------------	--

Return the argument with the smallest (minimum) value.

This function takes any number of arguments and return the argument with the smallest (minimum) value.

MOD	Num-var = MOD(Quotient, Divisor)
------------	----------------------------------

Return the remainder of the division between two numbers.

The **MOD** operator divides the two operands, *Quotient* and *Divisor*, and returns the remainder of

that division. The result of the initial division is truncated to an integer value, before the remainder is calculated.

RANDOMIZE	[number]
------------------	------------

Seed the random number generator.

number is a seed value. If *number* is not specified, a random value based on the TIME is used.

Values returned by the random number generator (**RND**) depend on an initial seed value. For a given seed value, **RND** always returns the same sequence of values, yielding a predictable pseudo-random number sequence. Thus, any program that depends on **RND** will run exactly the same way each time unless a different seed is given.

RND	Num-var = RND(From-value, To-Value)
------------	-------------------------------------

Return a random number.

If no operands are provided, **RND** returns a random value that is less than 1, but greater than or equal to 0. Numbers generated by **RND** aren't really random, but are the result of applying a pseudo-random transformation algorithm to a starting ("seed") value. Given the same seed, the **RND** algorithm always produces the same sequence of "random" numbers.

When From-value and To-value are provided, **RND (a , b)** returns a integer in the range of a to b inclusive. a and b can each be a numeric literal or a numeric expression.

ROUND	Num-var = ROUND(Num-expreression, Num-decimal-places)
--------------	---

Round a numeric value to a specified number of decimal places.

Num-decimal-places is an expression specifying the number of decimal places required in the result.

Rounding is done according to the "banker's rounding" principle: if the fractional digit being rounded off is exactly five, with no trailing digits, the number is rounded to the nearest even number. This provides better results, on average, than the simple "round up at five" approach.

A% = ROUND(0.5, 0) ' 0

A% = ROUND(1.5, 0) ' 2

A% = ROUND(2.5, 0) ' 2

A% = ROUND(2.51, 0) ' 3

SGN	Num-var = SGN(Num-expreression)
------------	---------------------------------

Return the sign of a numeric expression.

If *num-expression* is positive, **SGN** returns 1. If *num-expression* is zero, **SGN** returns 0. If *num-expression* is negative, **SGN** returns -1.

SIN	Num-var = SIN(Num-expreression)
------------	---------------------------------

Return the sine of its argument.

num-expression is an angle specified in radians. `SIN` returns a value between -1 and +1.

To convert radians to degrees, multiply by 57.29577951308232. To convert degrees to radians, multiply by 0.0174532925199433.

SQR	Num-var = SQR (Num-exprerssion)
------------	---------------------------------

Return the square root of its argument.

num-expression must be greater than or equal to zero. `SQR` calculates square roots using an optimized algorithm. That is, $y = \text{SQR}(x)$ takes less time to execute than $y = x^{0.5}$.

Attempting to take the square root of a negative number does not produce any run-time errors, but the results of such an operation are undefined.

Note that unlike some other languages, the square root function is `SQR` and not `SQRT`.

User Interaction

BEEP	
-------------	--

Emit a tone through the computer's speaker.

INPUTBOX\$	<code>Return-str = INPUTBOX\$(msg-str [, Title-str [, Default-str [, Use-Pass]]])</code>
-------------------	---

INPUTBOX\$ is used to prompt the user to input a value. The value is returned in Return-str. Note that except for the message string, all the other parameters are optional.

Msg-str is a string expression containing the message to show and prompt the user.

Title-str is a string expression containing the title of the message box. If omitted, the title will be ThinBasic.

Default-str is a string expression containing the default starting value of the answer box. If omitted, the default value will be an empty (null) string.

Use-Pass is a TRUE/FALSE flag to allow entry of password values. If you specify this as TRUE, your input will be obscured with * asterisks. This option is probably not of much use in SPFLite macros.

INPUTBOX\$ will allow you to click OK (or press Enter) or click Cancel. If you click on Cancel, the function will return an empty (null) string as a response, even if you typed something into the text box.

Note: If you use this function, you should consider using the [SPF Loop Check](#) function as well.

MSGBOX	<code>num-var = MSGBOX(hParent, Message [, Dialog-Style [, Title [, Timeout [, UpdateCountDown, [, CountdownFormat]]]]])</code>
---------------	---

Name	Type	Optional	Meaning
hParent	Number	No	Handle to the owner window of the message box to be created. If this parameter is zero, the message box has no owner window. Always code this as 0 (zero)
Message	String	No	A string expression that contains the message to be displayed.
Dialog-Style	Number	Yes	Specifies the contents and behavior of the dialog box. This parameter can be a combination of flags from the groups of flags allowed below for Dialog-Style.

Title	String	Yes	A string expression that contains the dialog box title. If this parameter is an empty string, the default title "thinBasic" will be used.
Timeout	Number	Yes	Number of milliseconds after which the message box will automatically exit with a %IDTIMEOUT return code.
UpdateCountDown	Number	Yes	Number of milliseconds time window for window caption count down update. If not present, no count down will be shown
CountDownFormat	String	Yes	Format of the count down. If not present a standard one will be used.

Display a Message Box for showing user any message and/or get user response.

NOTE: A call to MSGBOX can cause SPFLite to detect an apparent "loop" condition, because a macro that calls MSGBOX may wait indefinitely until the user clicks on OK or a similar button. You can control how SPFLite handles this situation by calling the function [SPF Loop Check](#).

NOTE: **thinBasic** supports the additional MSGBOX parameters: Timeout, UpdateCountDown and CountDownFormat. See the **thinBasic** documentation for more details. These options may not be useful in SPFLite macros.

Dialog-style is an equate to control the style of the box. It may be one of the following. These are numeric equate values. If you want more than one option, you can add options together. For example, to create a Yes/No box with NO as the default button, and a "warning" icon, you can specify the dialog style as %MB_YESNO+%MB_DEFBUTTON2+%MB_ICONWARNING. You can also use the OR operator instead of a + sign.

Symbol	Meaning
%MB_OK	The message box contains one push button: OK. This is the default. Because this style is the default, and the operand can be omitted, the shortest MSGBOX call you can write is MSGBOX(0,"message") .
%MB_OKCANCEL	The message box contains two push buttons: OK and Cancel.
%MB_YESNO	The message box contains two push buttons: Yes and No.
%MB_YESNOCANCEL	The message box contains three push buttons: Yes, No, and Cancel.
%MB_RETRYCANCEL	The message box contains two push buttons: Retry and Cancel.
%MB_ABORTRETRYIGNORE	The message box contains three push buttons: Abort, Retry, and Ignore.
%MB_CANCELTRYCONTINUE	The message box contains three push buttons: Cancel, Try Again, Continue.

%MB_DEFBUTTON1	The first button is the default button.
%MB_DEFBUTTON2	The second button is the default button.
%MB_DEFBUTTON3	The third button is the default button.
%MB_ICONEXCLAMATION	An exclamation-point icon appears in the message box.
%MB_ICONERROR	A stop-sign icon appears in the message box.
%MB_ICONINFORMATION	An icon consisting of a lowercase letter i in a circle appears in the message box.
%MB_ICONQUESTION	A question-mark icon appears in the message box.
%MB_ICONSTOP	A stop-sign icon appears in the message box.
%MB_ICONWARNING	An exclamation-point icon appears in the message box.
%MB_ICONASTERISK	An icon consisting of a lowercase letter i in a circle appears in the message box.
%MB_ICONHAND	A stop-sign icon appears in the message box.
%MB_APPLMODAL	<i>This option may not be useful in SPFLite macros</i>
%MB_SYSTEMMODAL	<i>This option may not be useful in SPFLite macros</i>
%MB_TOPMOST	<i>This option may not be useful in SPFLite macros</i>
%MB_HELP	Adds a Help button to the message box. When the user clicks the Help button or presses F1, the system sends a %WM_HELP message to the owner. <i>This option may not be useful in SPFLite macros</i>

num-var is a variable containing the user's response, which is useful if the message box contains more than one button to click on. It can contains one of the following equate values:

Symbol	Meaning
%IDABORT	ABORT button was clicked
%IDCANCEL	CANCEL button was clicked
%IDCONTINUE	CONTINUE button was clicked
%IDIGNORE	IGNORE button was clicked
%IDNO	NO button was clicked
%IDOK	OK button was clicked
%IDRETRY	RETRY button was clicked
%IDTRYAGAIN	TRYAGAIN button was clicked

%IDYES	YES button was clicked
%IDTIMEOUT	Timeout parameter was supplied, and MSGBOX timed out waiting for a user response

NOTE: If you use this function, you should consider using the [SPF Loop Check](#) function as well.

Created with the Personal Edition of HelpNDoc: [Maximize Your PDF Protection with These Simple Steps](#)

Appendix

Created with the Personal Edition of HelpNDoc: [Transform your help documentation into a stunning website](#)

Introduction to Keyboard Primitives

Contents

[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

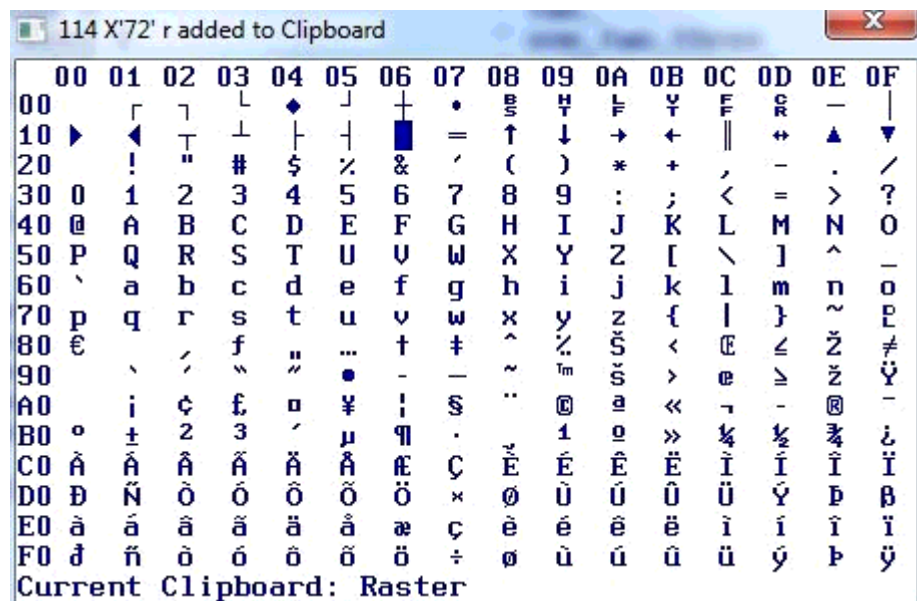
(BackSpace)	Move cursor left one position
(BackTab)	Move cursor left to previous tab stop
(BlockPaste)	Block-mode paste of clipboard contents
(Browse)	Use highlighted string as file name and open in Browse
(CharSet)	Display full 256 character set of edit font
(CharSetCol)	Same as above, but X / Y axes reversed
(ClipClear)	Clear the contents of the clipboard
(ClearInsert)	Clear both Inasert and DataInsert Mode and re-establish OVR mode.
(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
(CMarkDown)	Cond. Mark. If nothing currently marked, it acts like MarkDown, else ignored.
(CMarkLeft)	Cond. Mark. If nothing currently marked, it acts like MarkLeft, else ignored.
(CMarkRight)	Cond. Mark. If nothing currently marked, it acts like MarkRight, else ignored.
(CMarkUp)	Condi. Mark. If nothing currently marked, it acts like MarkUp, else ignored.
(CmdPad)	Insert the CmdPad.CLIP data into the Keyboard stream
(ClipTime)	Copy current time (Windows format) into clipboard
(CmdHome)	Move cursor to 'Home' based on current location
(CmdLog)	Toggle the Command Log function ON/OFF.
(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
(FMBack)	Move backward through the FM screen history.
(FMFwd)	Move forward through the FM screen history
(Home)	Move cursor to left-most position of primary command line
(Insert)	Toggle Insert/Overtyping 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
(MeditList)	Opens a popup with all loaded files in the MEdit session. Click on one to scroll directly to it.
(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
(PA2)	PA2 Attention, discard typing since last screen refresh
(PassThru)	Used so a key have its standard, Windows-defined behavior
(Paste)	Copy contents of clipboard to current cursor location
(Pen/colname)	Change color of selected text to the specified color
(Pen/Std)	Change color of selected text to standard text color
(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
(Refresh)	Refresh the File Manager display
(ResetInsert)	Turn Insert Mode off and return to Overtyping Mode
(RestoreCursor)	Position cursor where last (SaveCursor) was issued
(ResetCmd)	Reset outstanding Line Commands
(RestoreInsert)	Restore Insert/Overtyping 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
(TabBNDS)	Toggle Tab Bounds Mode
(TabRelease)	Will suspend Tabs handling when in TabBNDS mode.
(TabShift)	Shift data right to the 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
(Track)	Scroll the screen to the most recent Track Position
(TrackC)	Create a new Track Point at the current screen position
(TrackF)	Scroll the screen to the 'next' Track Position (i.e. Reverse of (Track))
(TxtHome)	Move cursor to column 1 of data line
(TxtNewLine)	Move cursor down one line then to column 1 of data line
(TxtNewLineNS)	Same as (TxtNewLine) 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, ...
- (ClearInsert)** This will clear all Insert and DataInsert status and re-establish normal **OVR** mode.
- (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.
- (CMarkDown)** Conditional Mark. If there is a current marked area, it is ignored, If nothing is marked, it will act like a normal (Markdown)

(CMarkLeft)	Conditional Mark. If there is a current marked area, it is ignored, If nothing is marked, it will act like a normal (MarkLeft)
(CMarkRight)	Conditional Mark. If there is a current marked area, it is ignored, If nothing is marked, it will act like a normal (MarkRight)
(CMarkUp)	Conditional Mark. If there is a current marked area, it is ignored, If nothing is marked, it will act like a normal (MarkUp)
(CmdHome)	Will selectively move the cursor based on it's current location. If in the text data area, it will move the cursor to the Line Command area. If anywhere else, it will move it to the Primary Command area
(CmdLog)	Will toggle (ON / OFF) the Command Logging trace. This may be requested by SPFLite support when resolving a reported program problem. It will create a log file of all command and keyboard primitive activity. The log will be date and time stamped and placed in the SPFLite HomeData folder. You may be requested to send this to SPFLite Support to assist in debugging the problem.
(CmdPad)	Will read the CmdPad.CLIP file and insert the contents into the Keyboard Stream. See CmdPad for more details.
(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)	<p>If there is currently highlighted, selected text on the screen, this will copy it to the clipboard.</p> <p>To copy data to a Named Private Clipboard, enter a / slash followed by the clipboard name. Example: (Copy / myclip)</p>
(CopyAdd) (CopyAdd / name)	<p>If there is currently highlighted, selected text on the screen, this will append it to the clipboard.</p> <p>(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.</p> <p>To copy data to a Named Private Clipboard, enter a / slash followed by the clipboard name. Example: (CopyAdd / myclip)</p>
(CopyLCmd) (CopyLCmd / name)	<p>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.</p> <p>Suppose you want the ability to 'propagate' a line command in a File Manager display. For example, suppose you want to edit several files.</p>

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) 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) 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) 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 toggle 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.

Data Shift Insert mode will also be reset if the normal (Insert) key is pressed.

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 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)
(Down/nn) Will move the cursor down 1 line if the optional **nn** parameter is not provided, otherwise it moves the cursor down the specified number of lines.

(Dup) 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.

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.

Example: before:

```
000001 line *** one of file
000002 line      two of file
000003 line /// three of file
000004 line      four of file
```

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:

```
000001 line *** one of file
000002 line  [highlighted] two of file
000003 line  [highlighted] three of file
000004 line  [highlighted] four of file
```

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:

```
000001 line *** one of file
000002 line *** two of file
000003 line *** three of file
000004 line *** four of file
```

(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" 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) (EnumHexLC) (EnumHexUC)	These primitives will trigger Enumerate processing. For a full description of this feature, see " Working with Enumerations ".
(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
(FMBack)	Will move the FM display backward through the recent history of File Manager Screens. This enables you to switch back and forth between recent FM display screens without having to re-select the various modes.
(FMFwd)	Will move the FM display forward through the recent history of File Manager Screens. This enables you to switch back and forth between recent FM display screens without having to re-select the various modes.
(Home)	Will move the cursor to the leftmost position of the primary command line.
(Insert)	Will toggle the current Insert/Overtyping 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.
(MeditList)	If you are in a MEdit session, this key will open a PopUp menu of all the files currently open in the MEdit session. You may then select, with a left mouse click, a file, and the session will be repositioned with that file at the top of the screen. See MultiEdit Sessions for more details
(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.
(PA2)	Triggers a PA2 (Program Attention 2) interrupt. There are times when you may have typed into the current screen unaware that the cursor was not where you thought it was. And perhaps you have overlaid text lines inadvertently. If you press the PA2 key BEFORE pressing any other Attention key (Like <i>Enter</i> , <i>PfFn</i> , <i>PgUp</i> , <i>PgDn</i> etc) then PA2 will 'throw away' all your keyboard activity since the screen was last displayed. Note: PA2 will internally do a RESET COMMAND function to clear any outstanding line commands. As long as you press no other keys, you can repeatedly toggle PA2 to go back and forth between the original and the modified version of the screen. If you do this Back-and-forth toggle, the reset line commands will not be restored.
(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.

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) The area of text data that is currently highlighted (via mouse or arrow keys) is displayed using the specified *colorname*. *Colornames* are specified in the Options -> Highlights 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) Will send a text form image of the entire edit screen to the Windows clipboard or Named Private Clipboard.

(PrtScrnClipboard / name)

(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) Will send only the actual data from the visible text lines to the Windows clipboard or Named Private Clipboard.

(PrtTextClipboard / name)

(Record [name]) 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. (If no *name* was provided, it will go to the normal Windows clipboard) You may then either issue CLIP *name* to edit the contents, or 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.

(Refresh) Can be used in the File Manager display to trigger a refresh of the contents. This can be needed if files are added or deleted by other external means.

(ResetCmd) This will reset any outstanding "Pending" line requests

(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.

- 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 sentence case, where the 1st word of each 'sentence' has its first letter capitalized. .
- (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.
(TabBNDs)	Will toggle Tab Bounds Mode On or Off. See "Working with Tab Bounds Mode" for a full Description.
(TabRelease)	Will suspend TAB handling when in TabBNDs mode. See "Working with Tab Bounds Mode" for a full Description.
(TabShift)	Will shift the data in the line from the current cursor position to the next defined TAB position in the TABS line. This is independent of whether TabBNDs mode is ON or OFF.
(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 title case, where the 1st letter of every word is capitalised.
(ToggleHome)	<p>If the cursor is currently on column 1 of the data, it will move to the leftmost non-blank character of text data.</p> <p>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.</p> <p>If the cursor is anywhere on a blank line, the cursor will move to column 1 of the data.</p> <p>If the cursor is located anywhere else, such as the primary command area or a non-data area, no action will occur.</p>
(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.
(Track)	This will move the screen back to the most recent Track Point .
(TrackC)	This will establish a new Track Point at the current screen position. This is mainly intended for use in a KB macro definition where the other commands may not create a Track Point automatically.
(TrackF)	This will move the screen back to the 'NEXT' Track Point . Typically used if you hit (Track) too many times
(TxtHome)	Will move the cursor to the leftmost visible column of the text data

(column 1 of the data).

- (TxtNewLine)** 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.
- (TxtNewLineNS)** Same as TxtNewLine but will not cause screen scrolling at the bottom of the screen.
- (Up)** Will move the cursor up 1 line if the optional *nn* 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 upper 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. For the purpose of this command 'words' are simple space delimited strings, not the full [WORD](#) specification used by FIND/CHANGE.
- (WordRight)** Will move the cursor left to the beginning of the next word. Repeated use will continue by moving to the next lines. For the purpose of this command 'words' are simple space delimited strings, not the full [WORD](#) specification used by FIND/CHANGE.

Supplied Fixed Fonts

Supplied Fixed Fonts

[Overview](#)

[AnonymousRegular](#)

[DejaVu Sans Mono](#)

[DejaVu Sans Mono Bold](#)

[JetBrainsMono](#)

[PixelCarnage Monospace](#)

[ProggyClean](#)

[ProggySmall](#)

[ProggySquare](#)

[QuickType Mono](#)

[Raize](#)

[Raster and Raster15](#)

[Raster14, Raster13 and Raster12](#)

[RasterTTF](#)

[Triskweline](#)

[IBM3270](#)

[IBMPlex Mono](#)

[MS LineDraw](#)

Overview

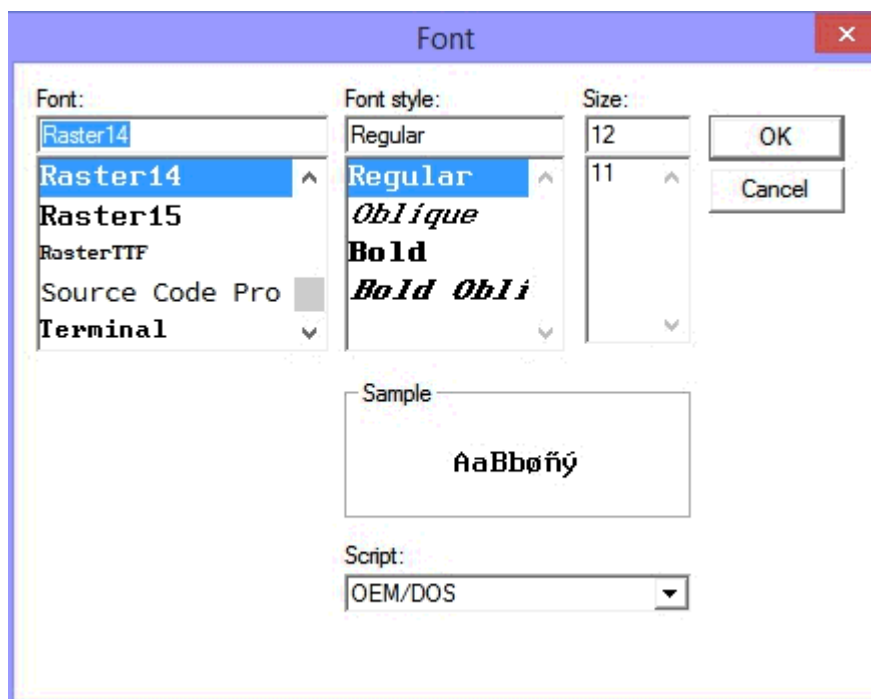
SPFLite can only use fixed pitch fonts for the edit screen. Although there are many free fonts available on the Internet, most of these are proportional fonts and are thus unusable. To supplement the standard Windows fonts like Courier and Terminal, the SPFLite website download page has an optional file 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:

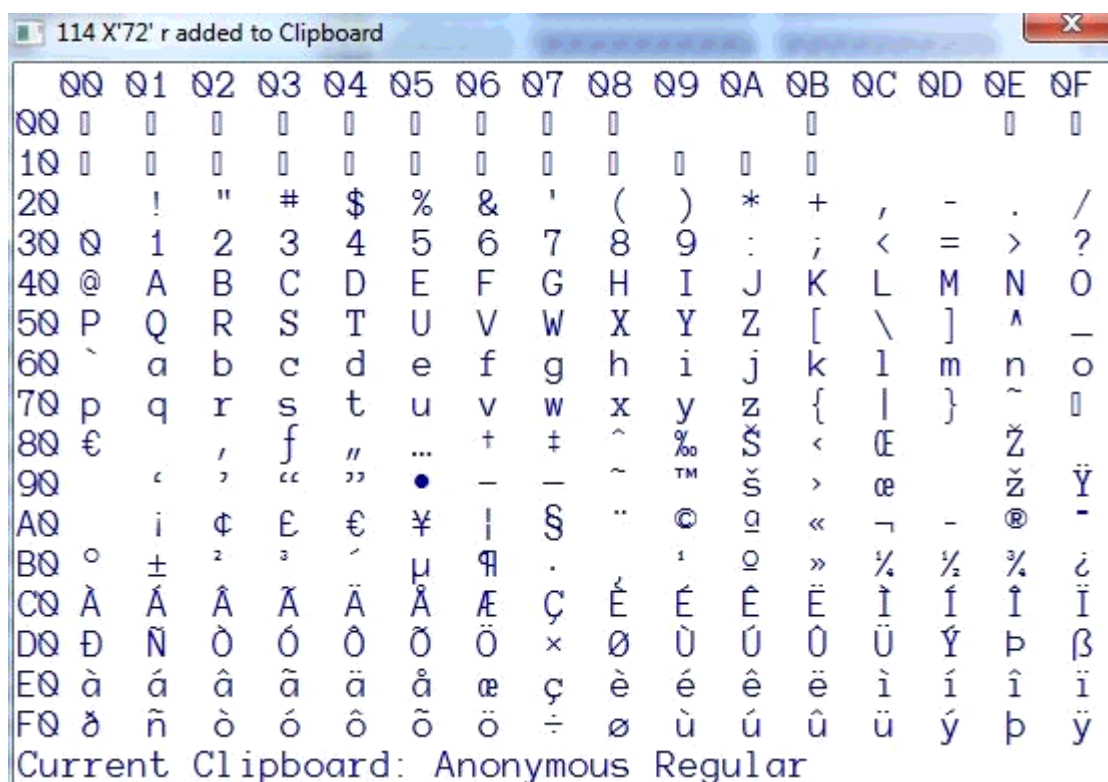


SPFLite causes only fixed-width fonts to appear in the font list.

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 ...																
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00																00
10																10
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	20
30	@	1	2	3	4	5	6	7	8	9	:	;	<	=	>	30
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	[\]	^	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	{		}	~	70
80	€	,	,	f	"	"	•	—	—	^	‰	Š	<	Œ	Ž	80
90		,	,	"	"	•	—	—	~	™	š	>	œ		ž	90
A0		ı	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	A0
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	B0
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	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	î	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:																
111	6F	'o'	16	"DejaVu Sans Mono"												

DejaVo 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																00
10																10
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	30
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	[\]	^	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	{		}	~	70
80	€			f	"	"	†	‡	^	‰	Š	<	œ		Ž	80
90		,	,	"	"	•	—	—	~	™	Š	>	œ		ž	90
A0		i	¢	£	¤	¥	¦	§	"	©	ª	«	¬	-	®	A0
B0	°	±	²	³	´	µ	¶	·		¹	º	»	¼	½	¾	B0
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	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	î	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"												

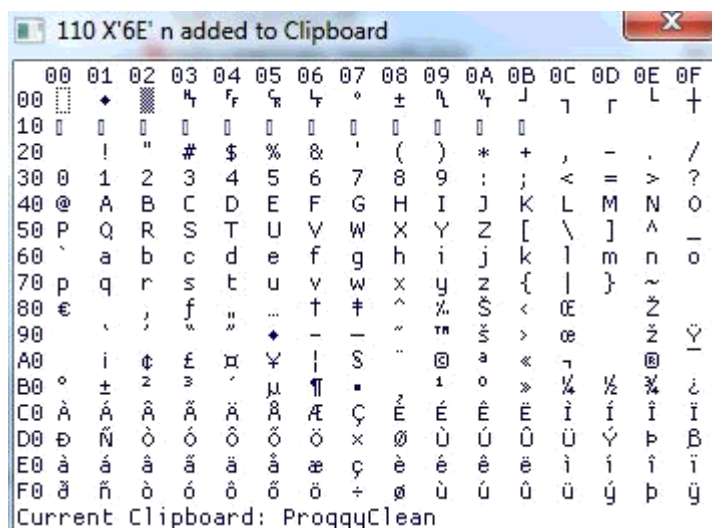
JetBrainsMono

ANSI - Left Click replaces Clipboard - Right Click adds to Clipboard																		
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	:	;	<	=	>	?	30	
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	40	
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	60	
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~		70	
80	€		,	f	"	...	†	‡	^	%	Š	<	œ		Ž		80	
90		,	,	"	"	...	-	-	~	™	š	>	œ		ž	ÿ	90	
A0		i	ç	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯	A0	
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	B0	
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	C0	
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	D0	
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	E0	
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	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' 13 "JetBrainsMono"																		

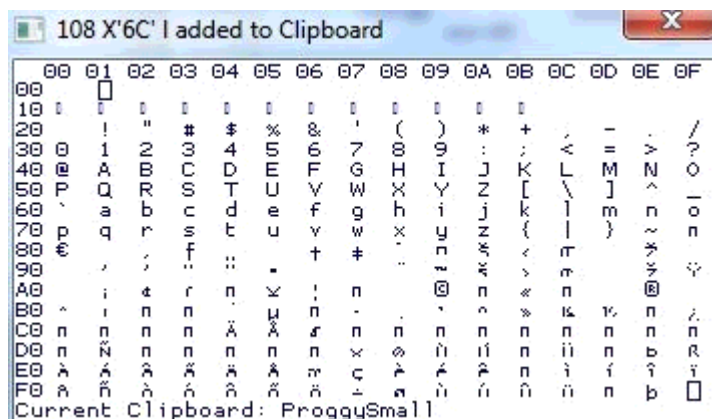
PixelCarnage Monospace

101 X'65' e added to Clipboard																		
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F																		
00	⓪	⓫	⓬	⓭	⓮	⓯	⓰	⓱	⓲	⓳	⓴	♂	♀	♂	♂	♂	00	
10																	10	
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20	
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	30	
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	40	
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	60	
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~		70	
80	ç	é	ê	ë	ä	å	ç	è	é	ê	ë	ì	í	î	ï	ð	80	
90		æ	æ	ô	ö	ö	û	ü	ü	ü	ü	ç	£		£	f	90	
A0	á	í	ó	ú	ñ	ñ	ë	ë	ë	ë	ë	¼	¼	¼	¼	»	A0	
B0																	B0	
C0																	C0	
D0																	D0	
E0	α	β	γ	π	Σ	σ	μ	γ	ξ	θ	Ω	δ	∞	∞	€	π	E0	
F0	≡	±	≥	≤	∫	J	÷	≈	°	.	.	√	n	z			F0	
Current Clipboard: PixelCarnage Monospace																		

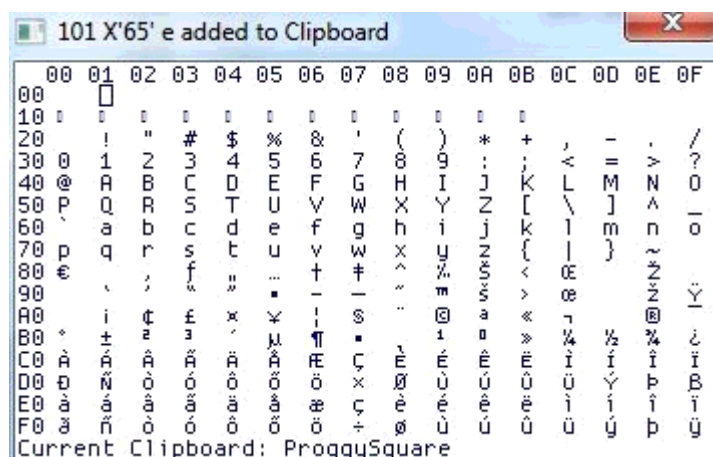
ProggyClean



ProggySmall

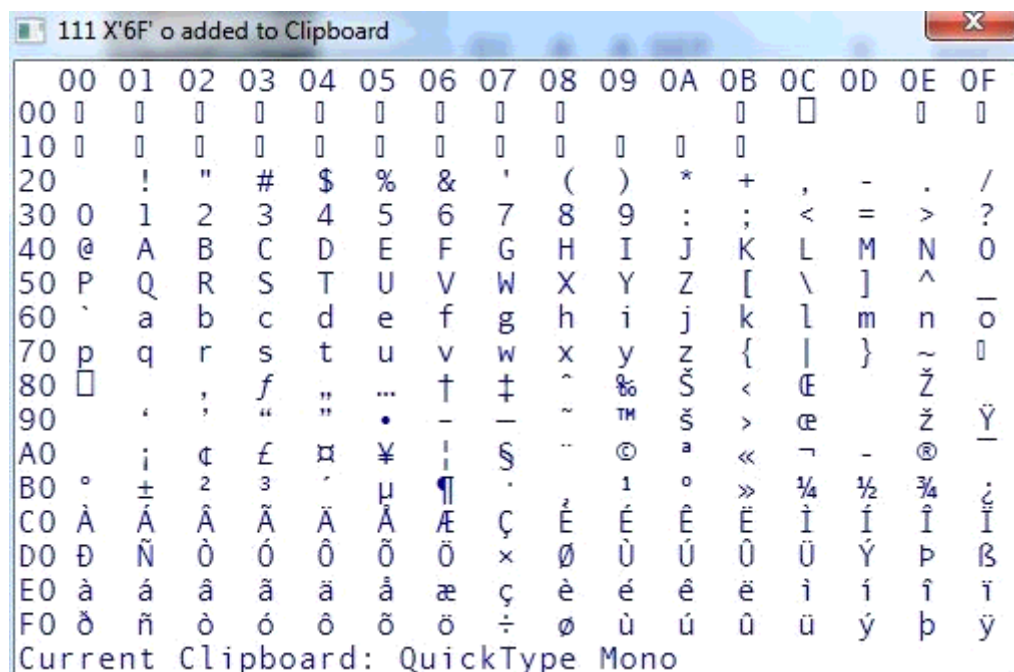


ProggySquare

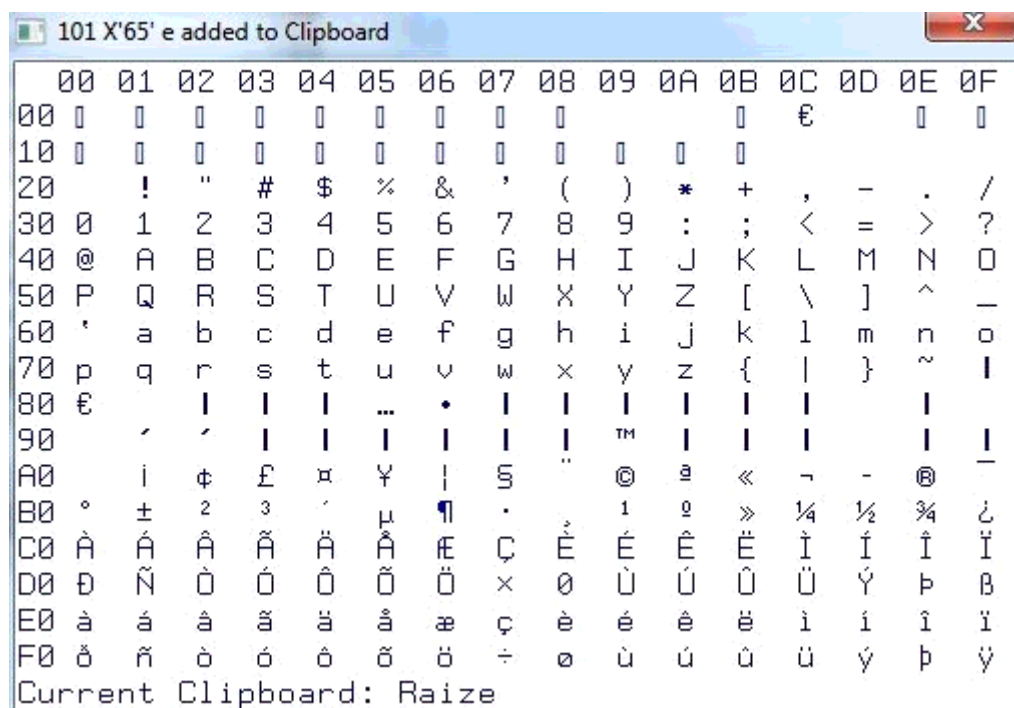


QuickType Mono

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.



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

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00			└	┐	↑	↓	→	←	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	00
10	0	+	()	4	5	6	7	8	9	*	+	,	-	.	/	10
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	30
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	40
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	60
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~		70
80																	80
90	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	90
A0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿		A0
B0																	B0
C0	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	C0
D0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿		D0
E0																	E0
F0	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	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

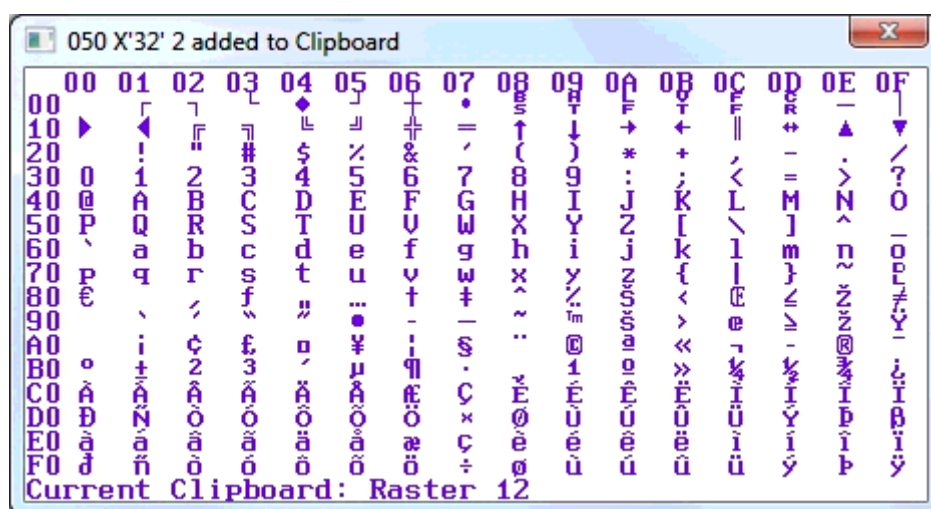
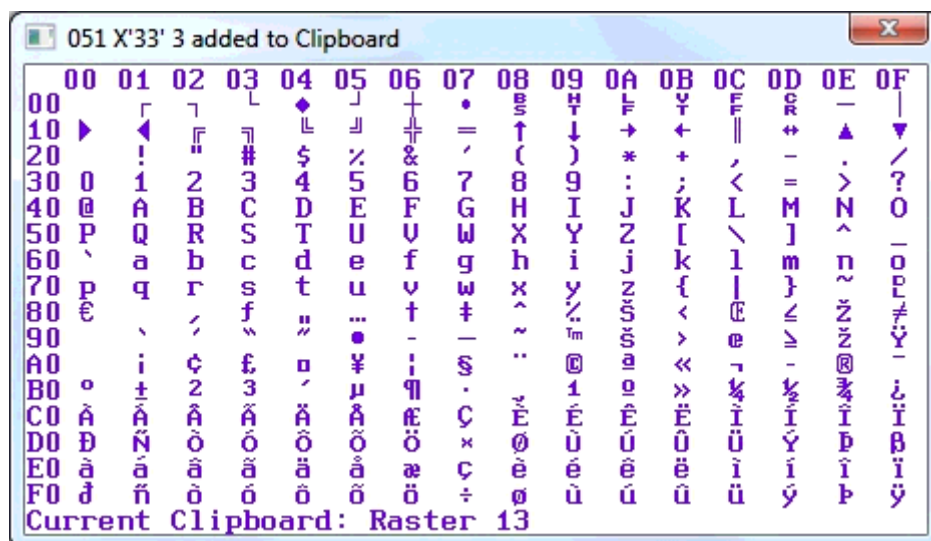
This is the same font as Raster/Raster15 but are shorter vertically (by eliminating white-space pixel rows) to allow a few more lines per screen.

052 X'34' 4 added to Clipboard

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00			└	┐	↑	↓	→	←	⌘	⌘	⌘	⌘	⌘	⌘	⌘	⌘	00
10	0	+	()	4	5	6	7	8	9	*	+	,	-	.	/	10
20		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	20
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	30
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	40
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	60
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~		70
80																	80
90	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	90
A0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿		A0
B0																	B0
C0	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	C0
D0	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿		D0
E0																	E0
F0	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	­	®	¯	°	F0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

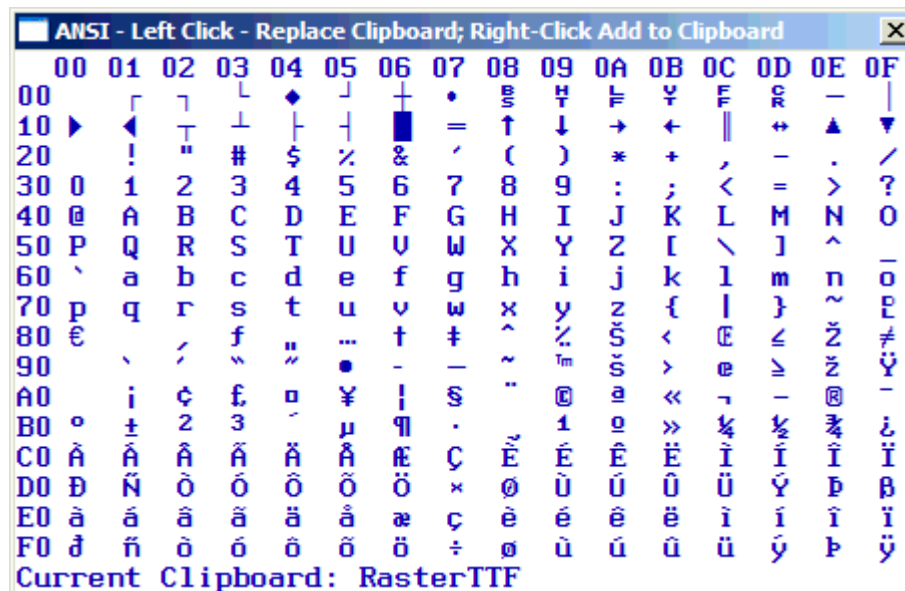
Current Clipboard: Raster14



RasterTTF

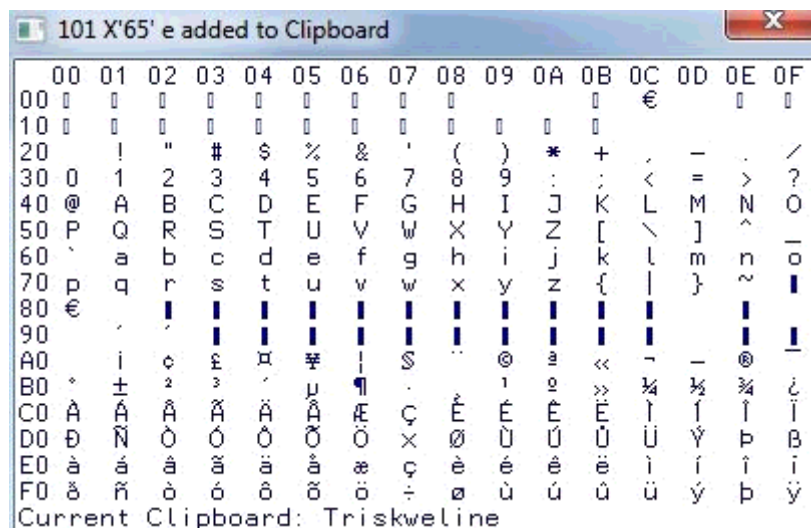
RasterTTF is the Raster font converted to TrueType format. RasterTTF has the same character set as Raster, but is intended primarily for printing files. RasterTTF has the advantage of being scalable, while Raster and Raster14 are crisper fonts for the screen. To print a mainframe-style "listing" file 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.

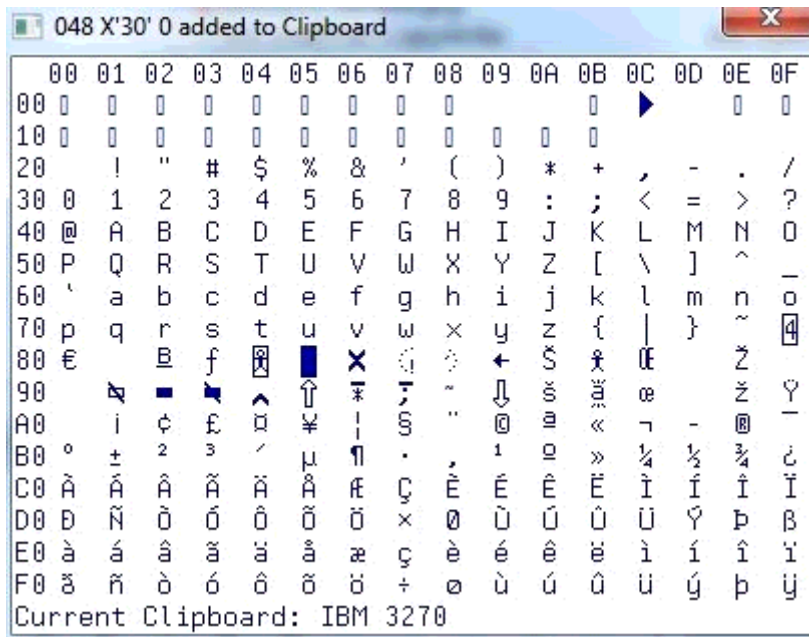


Triskweline

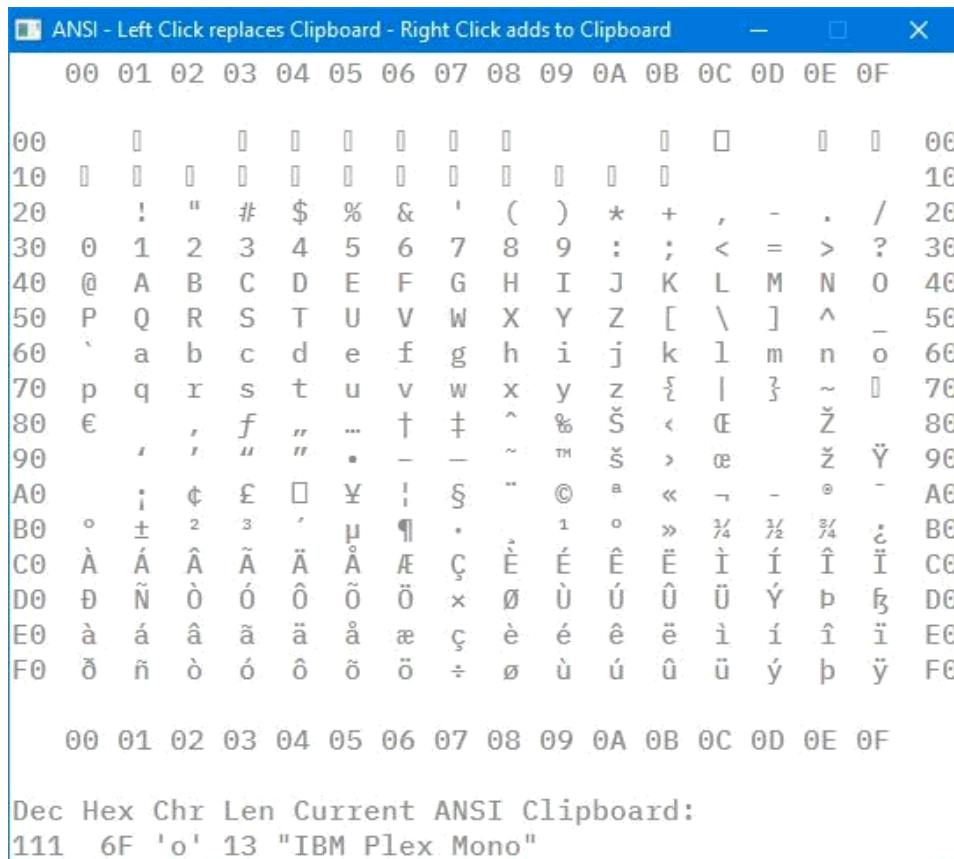
You are free to use this font, but don't ask us to pronounce it !



IBM3270



IBMPlex Mono



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	N	O
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	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
80							†	‡	ˆ							
90									˜							
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	¬	½	¼	;	«	»	
B0					†	‡	¶	§	¨	©	®	™	„	„	„	„
C0	Ł	ł	Ť	ť	—	†	‡	¶	§	¨	©	®	™	„	„	„
D0	Ł	ł	Ť	ť	—	†	‡	¶	§	¨	©	®	™	„	„	„
E0	α	β	Γ	π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	∅	ε	∩
F0	≡	±	≥	≤		∫	÷	≈	°	•	·	√	n	²	■	□

Current Clipboard: MS LineDraw

Automatic Colorization Files

Contents

[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 Profile selected by the [Extended File Type](#) process for the file being edited. Please review this topic to better understand how Profiles are chosen.

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. They were created 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.

AUTO files do not specify the actual [colors](#) to be used. The entries reference **SCHEME** numbers. The details of the colors SCHEME actually displays are specified in the [OPTIONS -> SCHEMES](#) dialog, which provides for 14 different color schemes.

Using schemes means if you are consistent in using scheme numbers for particular aspects of colorization (e.g. literals always request Scheme 7), then you can change the color of literals in ALL your AUTO files by simply editing Scheme 7 in [OPTIONS -> SCHEMES](#).

Being consistent here can also be assisted by the Alias support for Schemes. Review [OPTIONS -> SCHEMES](#) for more info. In the following descriptions, whenever a Scheme number is referred to, you can optionally substitute a Scheme Alias name.

Auto Capitalization Support

As well as colorization support, SPFLite can be requested to customise the capitalization all language keywords defined in the .AUTO file. For some computer languages, this can assist readability greatly. Note the capitalization 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.

Three options exist when identifying language Keywords. Each uses a unique AUTO command directive:

WORD

Identifies keywords which are to be colorized according to the specified Scheme number.

AUTOCAPS

identifies keywords which are to be colorized according to the specified Scheme number
AND
 which are to be capitalized.

AUTOCASE

identifies keywords which are to be colorized according to the specified Scheme number
AND
 the case of the word displayed is to be forced to agree with the case of the keyword as specified in the AUTOCASE statement.

e.g. If **AUTOCASE 12 %Yellow** is specified in the AUTO file, and the text **%YeLLow** appeared in the file, it would be displayed as **%Yellow**. This is convenient when CamelCase is used for many function or API references

When used, these options will cause all identified keywords in the associated .AUTO file to be case-modified as requested. The keywords are also processed as they are 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.

If you want the capitalization changes to be written to the file when saved, set the Profile [AUTOCAPS](#) to **ON**, the uppercasing will be 'finalized' and written to the output file. So what you **see** is what you **get**.

Note: The Profile [AUTOCAPS](#) command is a Primary command, although related, it is **not** the **AUTOCAPS** directive in the AUTO file described above.

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, Enter the OPTION CONFIG command to see the "Location for the SPFLite Data storage folders (MACRO, AUTO, STATE etc)". 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/AUTOCAPS/AUTOCASE 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)

NOTE: Comments must be on separate ; lines, comments **are not allowed** to be entered following the operands of a definition statement. The fact that they may sometimes **seem** to work should be ignored

- definition statements

Definition Statements

COMMENT_n sn start Up to Nine comment definitions are supported **COMMENT1** thru **COMMENT9**.
end

The **sn** operand refers to the scheme number used to display the comment. Each **COMMENT**_n statement **may** reference different scheme's if desired, or the same scheme. **sn** may be a hard-coded number, or a Scheme Alias name.

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 [xy]
[xy] ...

The **sn** operand specifies the **Scheme** number to be used for the display of quoted literal strings.

sn may be a hard-coded number, or a Scheme Alias name. If entered as simply QUOTED sn (with no optional parameters) then literal quoted strings are assumed to be delimited by the default set of Double quotes " "

You may specify a more exact 'set' of delimiters for quoted strings. The delimiters are entered as a pair of characters, separated by spaces. The first character is the 'opening delimiter' and the second character is the 'closing delimiter'.

Examples:

QUOTED 8 " "

This specifies that only double quotes are literal delimiters, the normal single quotes or back quotes will not be recognized as delimiters.

QUOTED 8 <>

This specifies that the less-than and greater-than characters are the literal string delimiters, none of the normal single, double or back quotes would be recognized as delimiters.

QUOTED 8 | | ' '

This specifies that only the vertical Bar | and single quotes will be used as literal delimiters, the normal single quotes or back quotes will not be recognized as delimiters.

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. ***sn*** may be a hard-coded number, or a Scheme Alias name. 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* AUTOCASE *sn string*

The **WORD / AUTOCAPS / AUTOCASE** 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. ***sn*** may be a hard-coded number, or a Scheme Alias name. Different **Schemes** 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.

When **AUTOCASE** is used instead of **WORD**, it requests forcing the the specified keyword to appear (case-wise) exactly as it is specified by the AUTOCASE statement.

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
COMMENT2 1  ::   1
; Delimiters
DELIMS      ( ) , . : = @
WORD        6  (
WORD        6  )
WORD        3  ,
WORD        3  .
WORD        3  :
WORD        3  =
WORD        3  @
;
;                               Reserved words
WORD        4  aux
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	(
)—Column Shift Right	Yes)
<—Data Shift Left	Yes	<
>—Data Shift Right	Yes	>
A—Specify an "After" Destination	Yes	A
B—Specify a "Before" Destination	Yes	B
BOUNDS—Define Boundary Columns	Yes	BNDS
C—Copy Lines	Yes	C/CC
COLS—Identify Columns	Yes	COLS
D—Delete Lines	Yes	D/DD
F—Show the First Line	Yes	F
I—Insert Lines	Yes	I
L—Show the Last Line(s)	Yes	L
LC—Convert Characters to Lowercase	Yes	LC/LCC
M—Move Lines	Yes	M/MM
MASK—Define Masks	Yes	MASK
MD—Make Dataline	Yes	MD
O—Overlay Lines	Yes	O/OO
R—Repeat Lines	Yes	R/RR
S—Show Lines	Yes	Not Supported
TABS—Control Tabs	Yes	TABS
TE—Text Entry	No	none - see note 1
TF—Text Flow	Yes	TF/TFE
TS—Text Split	Yes	TS
UC—Convert Characters to Uppercase	Yes	UC/UCC
X—Exclude Lines	Yes	X/XX

IBM Primary Commands

IBM Primary Command	Supported	SPFLite Command
AUTOLIST—Create a Source Listing Automatically	No	none - see note 2

AUTONUM—Number Lines Automatically	No	none - see note 3
AUTOSAVE—Save Data Automatically	Yes	AUTOSAVE
BOUNDS—Control the Edit Boundaries	Yes	BOUNDS
BROWSE—Browse from within an Edit Session	Yes	BROWSE
BUILTIN—Process a Built-In Command	No	none - see note 4
CANCEL—Cancel Edit Changes	Yes	CANCEL
CAPS—Control Automatic Character Conversion	Yes	CAPS
CHANGE—Change a Data String	Yes	CHANGE
COLS—Display Fixed Columns Line	Yes	COLS
COMPARE—Edit Compare	No	none - see note 5
COPY—Copy Data	Yes	COPY
CREATE—Create Data	Yes	CREATE
CUT—Cut and Save Lines	Yes	CUT
DEFINE—Define a Name	No	none - see note 4
DELETE—Delete Lines	Yes	DELETE
EDIT—Edit from within an Edit Session	Yes	EDIT
EDITSET—Display the Editor Settings Dialog	Yes	OPTIONS
END—End the Edit Session	Yes	END
EXCLUDE—Exclude Lines from the Display	Yes	EXCLUDE
FIND—Find a Data String	Yes	FIND
FLIP—Reverse Exclude Status of Lines	Yes	FLIP
HEX—Display Hexadecimal Characters	Yes	HEX
HIDE—Hide Excluded Lines Message	Yes	HIDE
HILITE—Enhanced Edit Coloring	Yes	HILITE
IMACRO—Specify an Initial Macro	Yes	IMACRO
LEVEL—Specify the Modification Level Number	No	none - see note 6
LOCATE—Locate a Line	Yes	LOCATE
MODEL—Copy a Model into the Current Data Set	No	none - see note 7
MOVE—Move Data	No	none - see note 8
NONUMBER—Turn Off Number Mode	No	none - see note 3
NOTES—Display Model Notes	No	none - see note 10
NULLS—Control Null Spaces	No	none - see note 11
NUMBER—Generate Sequence Numbers	No	none - see note 3
PACK—Compress Data	No	none - see note 12
PASTE—Move or Copy Lines from Clipboard	Yes	PASTE
PRESERVE—Enable Saving of Trailing Blanks	Yes	PRESERVE
PROFILE—Control and Display Your Profile	Yes	PROFILE
RCHANGE—Repeat a Change	Yes	RCHANGE
RECOVERY—Control Edit Recovery.	No	See OPTIONS -> General

RENUM—Renumber Data Set Lines	No	none - see note 3
REPLACE—Replace Data	Yes	REPLACE
RESET—Reset the Data Display	Yes	RESET
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: The Windows file system natively supports all of the ISPF compatible STATS information, except for the number of data lines in the file. A file's line count is stored in its STATE information when the file is saved, if STATE ON is in effect for that file type. If you wish to see the number of lines in a file when displaying it in File Manager, ensure that the file's PROFILE settings include STATE ON. If a given file has a PROFILE with STATE ON in effect but the line count does not appear in File Manager, you can use the L (Lines) FM line command to have SPFLite count the number of lines in the file, update the STATE information accordingly, and display the line count in FM. (The line count might be absent from the display if the file were stored into a directory but had not yet been directly saved by SPFLite.)

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

BOTTOM	BOT			
BROWSE	B	BRO	BR	
BOUNDS	BOUND	BND	BND	BOU
CANCEL	CAN			
CHANGE	C	CHA	CHG	
CHARS	CHAR			
COLS	COL			
COMPRESS	CP			
CREATE	CRE			
CURRENT	CURR			
DELETE	DEL			
EDIT	ED	E		
EXCLUDE	EXCLUDED	EXC	EX	X
FAVORITE	FAVOURITE	FAV		
FIND	F	FF		
HALF	H			
HILITE	HILIGHT	HI		
KEYMAP	KEYS	KEY	KBD	
LABELS	LABEL	LAB		
LOCATE	L	LOC		
LOCK	LOCKED			
MACRO	MAC			
MAX	M			
NDELETE	NDEL			
NFIND	NF			
NEXCLUDE	NX			
NREVERT	NV			
NULINE	NU			
OPTIONS	OPTION	OPT		
PAGE	P			
PREFIX	PRE	PFX		
PROFILE	PROF	PRO		
PTYPE	PT			
REPLACE	REPL	REP		
RECALL	RC			
RESET	RES			
REVERT	VV			
SAVE	SAV			
SPECIAL	SPE			
SUBMIT	SUB			
SUFFIX	SUF	SFX		
TABS	TAB			

TAGS	TAG
TOGGLE	TOG
ULINE	UU
UNLOCK	UNLOCKED
USING	USE
VIEW	V
WORDS	WORD

Created with the Personal Edition of HelpNDoc: [HelpNDoc's Project Analyzer: Incredible documentation assistant](#)
