FILE: GUIDEB

Compliments of FRESS

A File Retrieval and Editing SyStem

Release 9.1   2 MAY 79

FRESS Reference Manual

Structure and Commands

Andries van Dam

Carol L. Chomsky

Second Printing: January 25, 1919

0 PREFACE

This manual is not meant to be a self-teaching or exhaustive explanation of FRESS and thus not meant for those unfamiliar with FRESS at least on the level of the User's Guide. This guide is meant as a reference manual for those who have used FRESS extensively for editing and now want to avail themselves of some of its more sophisticated features and for those who want a complete, alphabetical listing of FRESS commands.

1 INTRODUCTION

In Section 6.1 of the FRESS User's Guide the term _structure_ was used in conjunction with remembering important points in the text and the creation of nonlinear "hypertexts." The purpose of the first sections of this manual is to elaborate on these concepts. In Section 2 below, a synopsis is given of the various structuring facilities within FRESS; Section 3 discusses alternative ways to view hypertext, and Section 4 contains some notes on how to edit structure. The alphabetical listing of _all_ FRESS commands of Section 5 also contains the appropriate functions for invoking these structure facilities.

2 HYPERTEXT

2.1 DEFINITION

Using FRESS as a simple text editor, the user can create a _linear_ piece of text (like an Egyptian scroll) with embedded format codes designed primarily to be an aid in preparing hard-copy documents. However, there are other features of FRESS which allow the user to add more complex, even _non-linear_ structure to his file. These features provide a means for producing more intricate documents; for example, one can use decimal block structure to allow dynamic renumbering of sections or chapters when they are moved (see Section 2.6); one can use keyworded jumps to allow selective printing of pieces of text (see Section 2.5).

In addition, one can use non-linear structure to create an on-line browsing environment. Related pieces of text can be linked together by cross-references (jumps, splices); comments, questions, and footnotes (annotations) can be added to the text without breaking the flow of the material as it is read; even graphical material can be added for viewing on a special display terminal.

The resultant "mobile" of text fragments is called a _hypertext_, "the combination of natural language text with the computer's capacities for interactive branching or dynamic display... a nonlinear text... which cannot be printed conveniently... on a conventional page...". A practical example of a hypertext might be an on-line encyclopedia or a set of programming and systems reference manuals, with each section in its own separate "area" (see Section 2.4) and with each cross-reference between area (topics) lightpen sensitive for potential selection. Functions are provided in FRESS to allow the fragments of text to be interpreted and examined in a variety of ways, in particular, to have linear paths traced through the hypertext either for online browsing purposes or for printing in a conventional manuscript form. The system also remembers the sequences of jumps that the reader has taken; this allows him to reverse his trail using the RETURN function.

2.2 ENCODING AND VIEWING STRUCTURE

In order to differentiate specially created structure (generically referred to as hypertext) from ordinary literal text, FRESS uses the "%" sign as a delimiter in displayed output. For example, the reader may remember that a label (e.g., "hypertext def") created via the Make Label (ML) command appears in the text as "%L(hypertext def)". The first character(s) following a % defining a piece of structure indicate what type it is. Thus the "L" here indicates that a special "location" or "point" is being created at this position in the text stream, while the parentheses and enclosed characters form the data associated with this point, in this case the text of the label. The generic format of a complete structure code thus is:

%<structure identification symbol><optional data>

The following table summarizes all possible structure as they appear in online display. The meaning and function of these structure types will be explained below.

location (point)	%L
block start	%<
block end	%>
jump	%J
pmuj	%P
splice	%SP
ecilps	%EC
tag	%T

The table below lists the various types of data fields, which pieces of structure they may be associated with, and how they are displayed online. See the explanations for individual commands for information on how to create and edit these data fields.

Data Field Type	How Displayed	May be attached to:
Dec lab number	within single quotes	dec lab ref tags, dec blocks
Label	within parentheses	points(locations), block starts
Viewspecs	within parentheses	jumps/pmujs, splices/ecilpses
Keyword	within double quotes	annotation tags, blocks, jumps/pmujs, splices/ecilpses
Explainer	followed by %%	jumps/pmujs, splices/ecilpses
Picture name	within single quotes	picture reference tag

If more than one data field appears on a particular piece of structure, they appear in the order indicated in the table. For example, a decimal block with label and keyword data fields would appear as:

```
%< '1.3' (label)"keyword"text in block%>
```

A jump with viewspec, keyword, and explainer data fields would appear as:

```
%J(print)"keyed"this is the explainer%%
```

The appearance of structure is important so that the reader may recognize it in the text and because of the specialized rules which apply to editing of structure (see Section 4). Note that, unlike format codes, none of this structure can be entered as part of literal text input. Because of the fact that FRESS maintains complex control information, specialized commands such as Make Jump, Make Splice, etc. must be used.

2.3 LABELS AND LOCATIONS (POINTS)

Labels are not actually a separate kind of hypertext structure. They are merely a type of data field which can be attached to a piece of structure, in particular to locations (also called points) and block starts (see Section 2.6). However, they most often appear as the only data field of a location and are used as a means of random access to a particular place in the file. They are made using the Make Label (ML) command and retrieved with the Get Label (GL) command. (They can also be made as a data field to a block using the Make Block (MB), Insert Block (IBL), Make Decimal Block (MDB), and Insert Decimal Block (IDB) commands. A label can be placed on an existing block start using the Make Label command and specifying the block start as the <lp>.)

All labels in the file appear in alphabetical order in the label space which can be displayed by using the Display Space (DS) command. (See Section 3.2.5 for detailed information about the label space.)

2.4 AREAS

Sometimes the user desires to segment his text into independent, arbitrarily long fragments called text areas. Each area is a continuous linear string of text and might be a chapter, an entire book, or a short footnote. When a file is created, the first such area in the text space is created and all text goes inside it. Other areas may be created using the Make Area (MA) and Split Area (SP) commands.

Each area is delimited by *START OF TEXT AREA* and *END OF TEXT AREA* lines. These are considered structure although they do not start with percent signs as do all other pieces of structure. FRESS does not allow the user to scroll backwards beyond the start line or forwards beyond the end line. Therefore, the user should either have a label inside each area or a jump to each area from another area in order to get there (except, of course, the first and last area, which may be accessed by the Display Space (DS) and Bottom (B) commands). Although the user may not scroll between areas, a printout of a file using the Fullprint command will ignore area lines.

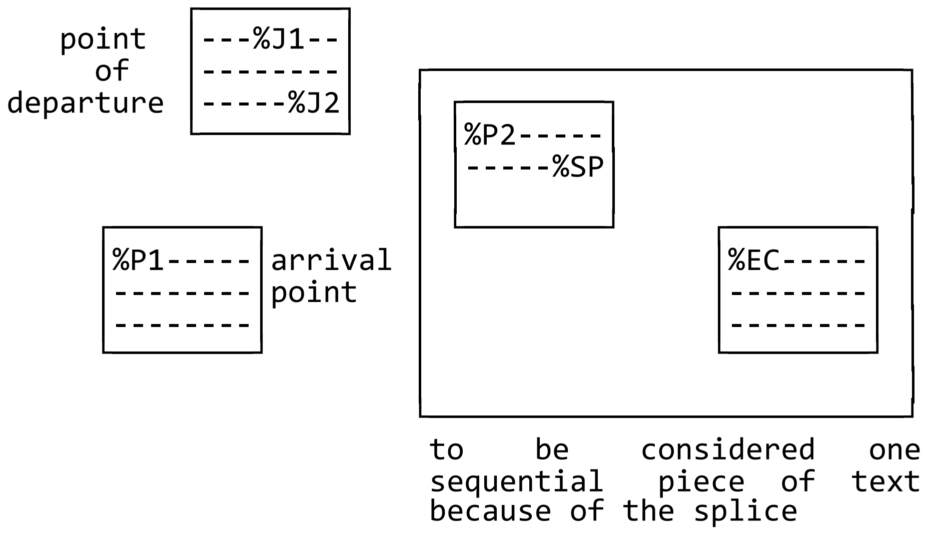
Multiple areas are useful to conveniently separate unrelated pieces of text. Keeping material in separate areas rather than separate files has the advantage of allowing easier keyword and label retrieval.

2.5 JUMPS AND SPLICES

Jumps are conditional links between two text sections; they are used primarily in a display console, browsing-oriented environment to provide online indexes, or optional references to additional details, explanations, etc.

Unlike the jump, a splice unconditionally grafts together sections of text such that they follow one another without user action when the file is displayed online or printed. A splice can be used to "splice around" unwanted sections of text or perhaps to concatenate files or areas.

The following is a schematic representation of the relationship between text fragments linked by jumps and splices.



2.5.1 JUMPS VS. LABELS

One major difference between jumps and labels is that the latter can be accessed (retrieved) regardless of where the user is positioned in his file (i.e., random access ability) while the former are in evidence only when the user encounters the point of departure. In a typewriter oriented system used principally for editing, jumps would probably be used only in their keyworded form (see below) for conditional file linking, while labels would be used heavily for online retrieval. Note also that labels are attached at a single unique location in the text while jumps link two locations.

2.5.2 PMUJS AND ECILPSES

Both jumps and splices have inverses (called "pmuj" or backward jump, and "ecilps" or backward splice respectively) to allow bi-directional travelling. When a jump is created with the Make Jump (MJ) command, a jump/pmuj pair is actually created. Similarly, Make Splice (MS) creates a splice/ecilps pair. No distinction is made between jumps and splices and their inverses, except that the user may tell them apart visually as explained below.

2.5.3 KEYWORDED JUMPS

When a jump or splice is created, the user may specify a string of one or more keywords to be associated with the structure. When a keyworded jump is encountered while displaying or printing, this string is compared to the "jump keyword request string", a "boolean request" previously set using the Set Keyword Jump Request String (SKJUMP) function. If the request is satisfied by the jump's keywords (see Section 3.1), the jump is treated as if it were a splice. This feature provides an easy way to conditionally splice text areas or files, thereby allowing a user to automatically follow any of a number of distinct paths through his text.

2.5.4 VIEWSPECS

Viewspecs (see Section 3.3) on any of these structures (jumps, pmujs, splices, and ecilpses) replace the user's current viewspecs if the link is taken (explicitly by the Jump command, implicitly as a splice, or because of a keyword request). The new viewspecs take effect at the point to which the link goes. The viewspecs that were in effect before the link was taken are saved by the system so that they can be re-invoked by RETURN or when the splice is returned to by scrolling back over the ecilps. If a link is not taken its viewspecs are ignored. The viewspecs on a pmuj are specified separately in the Make Jump from those on the jump and therefore can differ.

2.5.5 JUMP AND SPLICE CREATION AND VIEWING

Created by the Make Jump (MJ) command, a jump and its corresponding pmuj appears in the text with an "explainer", a user designated description of the location to which the user may jump, and optionally, viewspecs and keywords. Splices and their corresponding ecilpses are similarly created with their explainers with the Make Splice (MS) command. Explainers must be less than 256 characters in length and can be edited once they are created.

To view ("take") a previously created jump, the online reader specifies the Jump (J) command and light-pens (LP's) a jump or its explainer; FRESS will then display the jumped-to place in the text. If the user wishes to return to his original location, he may use the RETURN command, or he may use the Jump command again, this time specifying the "pmuj" or its explainer. Splices and ecilpses, as explained above, are "taken" by being scrolled over without explicit user action, but the explainers and the splice symbol (as explained below) serve the function of alerting the reader. If the user has scrolled forward over (taken) a splice or (satisfied) keyworded jump, scrolling backward over the ecilpse (or keyworded pmuj) will return the user to the splice (i.e., will take the ecilps). However, scrolling backward over an ecilps (or keyworded pmuj) which has not been reached by scrolling forward over the other end will cause no such non-linear travel.

As an example, consider a file set up as follows:

```
         •
         •                      text
         •                        •
    1   text                      •        2
        %SP                       •
                                %EC
                                  •
                                  •
                                %L(lab)    3
```

If the user scrolls from point 1 to point 3 across the splice and then scrolls backwards, he will return across the splice to point 1. If, instead, he scrolls from point 2 to point 3 across the ecilps and then scrolls backwards, he will return to point 2, not taking the ecilps. Thus in either case, the user will follow the same path forward and back. If the user arrived at point 3 by some non-linear path, perhaps a Get Label, and then scrolled backward, he would not take the ecilps and thus would reach point 2.

When a splice or keyworded jump is 'taken', the jump and pmuj or splice and eclips are displayed consecutively, separated by a display jump symbol (&). Thus a completed splice might look as follows:

```
%SPexpl1&%ECexpl2
```

2.6 BLOCKS

Jumps and splices (and labels as well) occur at single, clearly defined locations in a text file. Such _points_ have no scope, i.e., they have significance only at the position in the text where they appear.

Sometimes it is useful to associate control information with an entire section of text, consisting of one or more characters. For example, one typically thinks of applying keywords or section numbers to such a "block" of text rather than to a single point in the file. A block, then, is a string of text of arbitrary length enclosed (delimited) by a _block start_ (%<) and a _block end_ (%>).

2.6.1 KEYWORDED BLOCKS

Blocks by themselves are useful only in making it easier to move large pieces of text as single units. Another more important use is to assign keywords to each block to indicate information about the contents of that block. Then special FRESS commands can be used to retrieve sets of related blocks for online reading or offline printout. These commands (Block Trail Continuous (BT) and Block Trail Discrete (BTD)) create "block trails", that is, a sequence of blocks all of which have keyword strings satisfied by the "boolean request" given in such a command. Thus on the basis of the keywords which the user assigns to his blocks, the system can isolate a subset of information on a given subject. Section 3.1 covers the various rules governing keyword requests of all types.

Additionally, to allow easy access to <u>single</u> unique blocks, labels may be assigned to them.

2.6.2 DECIMALLY LABELLED BLOCKS

Decimal blocks are simply blocks with numbered labels associated with them. They provide a means for superimposing an outline form on the text in the file. A typical outline form using decimal labels would be

```
1 Introduction
2 Oil resources
  2.1 Estimates of reserves
  2.2 Development
    2.2.1 Extraction processes
    2.2.2 Costs
  2.3 Distribution
    2.3.1 Refining
    2.3.2 Transportation
  2.4 Consumption
3 New supplies
4 Conclusion
```

This structure is superimposed on a file by putting the text in each section inside a decimal block. A block contained in another block is said to be nested in that block, e.g., 2.2.1 and 2.2.2 are nested in block 2.1, which is in turn nested in block 2. The level of a block is determined by how deeply it is nested inside other blocks. For example, 1, 2, 3, and 4 are on level 1; 2.1 and 2.2 are on level 2; 2.2.1 and 2.2.2 are on level 3. Decimal blocks appear in the text as regular blocks with the decimal label enclosed in apostrophes as a data field of the block start. For example

```
%< '2.1' Estimates of reserves%>
```

Nested blocks begin new lines, as in the outline above.

Decimally labelled blocks are created with the Insert Decimal Block (IDB) and Make Decimal Block (MDB) commands, the former being used to input new material into the newly created block, the latter to enclose an already existing text string in a block.

Decimal label numbers are generated dynamically by FRESS to accomodate changes in the decimal structure caused by editing; they are not part of the text stored in the file. For this reason the number can not be edited (see Section 4.1.3 for an exception). The number of a decimal block depends upon its position with respect to other decimal blocks, calculated from the start of the file. In the main text space this includes any rearrangement that results from splices or keyworded jumps taken within a single file.

For example, consider a file with the following structure:

```
      •
      •
%< '1'    %>
%< '2'    %>
%J"key"
%< '3'    %>
%P"key"
%< '4'
%< '4.1'    %>%>
      •
      •
```

If the user set up a boolean request string using the Set Keyword Jump Request String (SKJ) command to satisfy the above jump, the structure of the file would now appear as:

```
      •
      •
%< '1'    %>
%< '2'    %>
%J"key"&%P"key"
%< '3'
%< '3.1'    %>%>
      •
      •
```

The numbering of the blocks is automatically changed to reflect the "disappearance" of the decimal block appearing third geographically when the jump was not taken.

An additional difficulty arises here, however. If the user sets his keyword request string as in the second case above, then does a Bottom and scrolls backwards, he will not take the keyworded pmuj and thus will encounter the decimal block which does not appear in the second case above. However, counting is done from the top of the file and the keyworded jump <u>is</u> taken when going forwards; therefore the number of the block which is spliced out when travelling forwards will be incorrect in this case. Any blocks spliced out in this manner will be displayed as having the same number as the first block after the ecilps or keyworded pmuj.

In addition to following splices and keyworded jumps within a single file, the printout of a file made with the Fullprint (FU) command will continue sequential numbering if an interfile jump or splice is taken. This facility is not yet available in online display.

Decimal level nesting is currently limited to eight levels. The number of blocks on each level is limited to 8,191.

Decimal labels can be used like labels to retrieve single blocks with the Get Decimal Label (GDL) command.

The Fullprint of a file treats decimal block starts as heading codes. The text inside the block up to the next new line format code is used as the body of the heading. The body of the block should therefore always begin with a new line format code so as to prevent its inclusion in the heading. The heading number is the same as the nesting level of the block.

<u>2.6.2.1 Decimal blocks in the structure space</u>

Unlike in the main text space, the dynamic numbering of decimal blocks in the structure space does <u>not</u> follow splices and keyworded jumps. The structure space is meant to be a linear representation of <u>all</u> the non-linear structure of the file, and thus includes even those structures which are 'spliced out' in the main text, work or annotation spaces. As a result, the decimal block numbers of the blocks in the structure space will not always match the numbers of the corresponding blocks in a main space. It is therefore wise to put labels or keywords on decimal blocks to help identify them uniquely.

2.6.2.2 Editing decimal blocks

A block in general, and a decimal block in particular, is considered a unit which must be manipulated as such. Thus the user can move an entire block, including the text contained inside it; he cannot move the block start or the block end alone in order to expand or contract the size of the block. Thus if the user had a file which had two consecutive decimal blocks:

```
%< '1' text1%>
%< '2' text2%>
```

and wished to move block 2 so that it would be nested in block 1 and therefore become block 1.1, he could not do it by moving the block end of block 1 to follow the block end of block 2. He also may not change the location of block 2 by editing its decimal number, since the numbers are determined by the position of the blocks, not vice versa. The proper procedure is to move the block to follow the last piece of text in block 1. Thus he might type

```
m/%< '2'...%/text1
```

For an easier way of specifying this command, see Section 5.3 on Command Qualifiers.

The user must also remember that any rearrangement of decimal blocks in one part of a file may change the numbering of decimal blocks later in the file. For this reason it is useful, as mentioned above, to use labels or keywords on decimal blocks. If the user plans to do complex manipulations on decimal blocks, it is helpful to work on these from the bottom of the file to the top so that the numbering of a block does not change until after the editing has been done on that block.

2.6.2.3 Printing decimal blocks

An option has been added to the Fullprint command which allows the user to specify the action taken when a particular decimal block level is reached. The default is to indent the text in a nested block 3 more spaces than the containing block. Thus the text in block 1 would start at the current margin, the text in block 1.1, 1.2, etc. would start 3 spaces to the right, etc.

To change this, specify 'M' as an option when FU'ing, i.e.,

```
FU M
```

This causes a search of the file for a series of macro definitions, two for each decimal level encountered. These macros are !.dm_m_. and !.h_n_. where _n_ is the decimal label level, and _m_=_n_-1. The !.dm_m_. macro should be defined as the desired margin setting for that block, using the !+MARGIN+ alter code. The !.h_n_. macro should be defined as a heading code (!-hn-).

For example, the default macro definitions are

```
!.dm0=!+margin0,0+.      !.h1=!-h1-.
!.dm1=!+margin3+         !.h2=!-h2-.
!.dm2=!+margin6+.        !.h3=!-h3-.
       etc.                 etc.
```

In addition, heading levels can be redefined as desired. This should be done very carefully, since no error checking is done when the file is printed using Fullprint. Changing heading level meanings is done using the !+HEAD+ alter code. The format is

```
!+HEADn=a,b,c;n=. . .+
```

where n is the heading level being redefined,
a is the sum of any combination of the following:
- 4 heading title is in caps
- 32 heading title is underscored
- 1 new page before heading
- 0 nothing done

b is the number of lines to skip before the heading
c is the number of lines to skip after the heading.

For example, to define heading level 1 so that the title will be in caps and underscored, with two lines skipped before and three lines after the heading, the file should contain

```
!+HEAD1=36,2,3+
```

2.6.2.4 Decimal Label Reference Tag

Another feature useful along with decimally labeled blocks is the decimal label reference tag. This allows the user to refer to one or more of his decimal blocks by number in the text and have the number be dynamically updated depending upon the nested level of that block. Thus, a line of text such as "see Section 3.2" could appear on the Fullprint by having the user place a tag in the file instead of the actual decimal number; if the decimal label level of that particular block is changed, the tag will automatically reflect this in online display and Fullprint. In online display, a decimal label reference tag appears as

```
%Tƀ'dec#'ƀ
```

where dec# is the decimal number of the block referred to. (ƀ indicates a blank.) In the Fullprint of a file, only the decimal number appears. (See Make Decimal Reference (MDR), and Make Decimal Reference Deferred (MDRD)).

2.6.2.5 Implied Insert Stack

To facilitate the insertion of blocks, FRESS maintains what is known as an implied insert stack. This is a stack of implied insert points used for both creating blocks and for ordinary edits, just as any other implied insert points. They are discussed in this section because they are particularly useful when handling decimal blocks.

Each entry in the stack is a pointer to the position where the next block will be inserted if no specific LP is given in a Make Block (MB), Make Decimal Block (MDB), Insert Block (IBL), or Insert Decimal Block (IDB) command.

Two entries are made in the stack when any of these four commands are used: one points at the character before the block end, the other at the last character of the block end of this new block. A new block can thus be made easily inside (nested in) or immediately after (same level as) the previous one. The top pointer in the stack, that is, the current implied insert point, will be the first pointer, i.e., the pointer at the character before the block end of the block just inserted or created.

The first examples below will be given using the Insert Decimal Block (IDB) command. The format of this command (see Section 5.1) is

```
     IDB   <point   to   make   block  after>°  <label>°
          <keywords>° <text>
```

The last parameter specified when the command is given is assumed to be the <text> to be inserted inside the new block. All other parameters are assigned left to right unless specifically left null. For example, in the command

```
     IDB//lab/words
```

the <point to make block after> is specified null. 'lab' is to be a label on the block, and 'words' is the text string to be inserted (since the <keywords> parameter is optional and <text> is not). If the user, instead, desired to insert 'words' as the keyword string and go into input mode to add the text to go inside the block he would type

```
     IDB//lab/words/
```

leaving the <text> parameter purposely null to enter input mode. When the <point to make block after> is left null, as in both examples, the implied insert pointer is used.

The following example explains the use of the implied insert stack in manipulating decimal blocks. The command

```
     IDB/<location>/Introduction
```

would result in the following text in the file:

```
     %< '1' Introduction%>
```

The implied insert point is after the word 'Introduction'. An ordinary editing command making use of the implied insert point, such as

```
     I/word
```

would result in

```
     %< '1' Introductionword%>
```

If, instead, the command

```
     IDB/word
```

were given, it would result in:

```
     %< '1' Introduction
     %< '1.1' word%>%>
```

A decimal block has been inserted at the implied insert point. Note that the block inserted is nested, that is, contained in, the previous block (cf. the %>%> indicating 2 consecutive block ends). If, instead, the command

```
     IDB/
```

were given, leaving both the insert point and text parameters null, a decimal block would be created at the implied insert point, as before, and the user would be placed into Input Mode inside that block. In either case, the new implied insert point is at the end of the text in block 1.1. If a new block is inserted using this implied insert point it will be block 1.1.1, nested inside the previous block, 1.1.

The file would now look like:

```
     %< '1' Introduction
     %< '1.1' word
     %< '1.1.1' text inside a block%>%>%>
```

The implied insert point is after the word 'block'. Another block inserted in the same manner as above would result in a block numbered 1.1.1.1. If, however, the user does not wish this next block to be nested inside the previous block, but rather to be on the same level as the previous block, he executes the command &POP (or &P). This 'pops' the implied insert stack to the next outermost level of decimal blocks. Thus, in the example above, if the implied insert point is after the word 'block' and the command &P is given, the new implied insert point is after the string:

```
     block%>
```

If, then, the command

```
     IDB/new one
```

were given, the result would be:

```
%< '1' Introduction
%< '1.1' word
%< '1.1.1' text inside a block%>
%< '1.1.2' new one %>%>%>
```

The implied insert point would then be after the word 'one'. If another block on the same level as 1.1.1 and 1.1.2 were desired, the sequence of commands would be:

```
&P
IDB/this text will be on level 1.1.3.
```

Similarly, if the next block to be inserted is to be on the same level as 1.1, two &P commands must be given. The first will bring the implied insert point to the same level as 1.1.3, the second to the level of 1.

The following example, using non-decimal blocks, shows the position of the implied insert pointer as pointers are created and popped. Note that each entry shows the User command, the System response, and the position of the implied insert Pointers in the implied insert stack, indicated here with numbers denoting their position in the stack.

original string:

```
"...text in old file..."
```

```
U:  ib/file/text inside the first block
S:  %<text inside the first block%>
P:                              1 2


U:  mb/inside
S:  %<inside%> the first block%>
P:         1 2               3 4


U:  i/ the second block
S:  %<inside the second block%> the first block%>
P:                          1 2               3 4


U:  &P
S:  %<inside the second block%> the first block%>
P:                            1               2 3
```

```
U:  i/popped textø
S:  %>popped text the first block%>
P:                 1             2 3


U:  &P
S:  %> popped text the first block%>
P:                                1 2


U:  &P
S:  %>popped text the first block%>
P:                                1
```

The stack of pointers is currently limited to eight pointers at one time or four successive MBLOCK commands with no intervening &P's. If a block is made when the stack is full, the pointers from the oldest block are dropped off the bottom of the stack and the new pointers are added normally to the top.

2.6.3 ANNOTATION BLOCKS

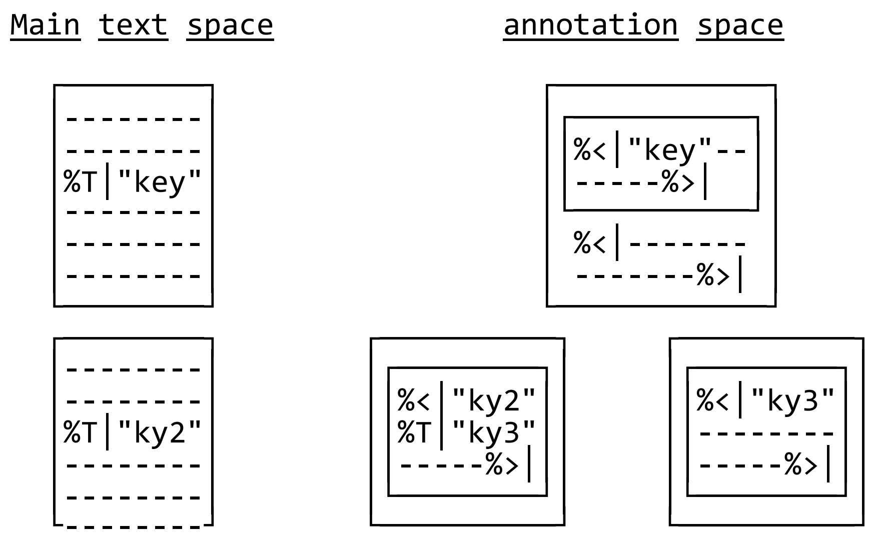
There are certain types of additions to text material -- comments, questions, references, etc. -- which the user might wish to keep separate from the main body of the text while allowing retrieval of that information at any time, either automatically or by choice. It was for this purpose that the FRESS facility for 'annotations' was developed. Using the Insert Annotation (IA) or Make Annotation (MA) commands, the user can cause a piece of text to be filed away in a separate section of the file called the annotation space. These pieces of text can then be referenced at relevant places in the main text. These references take the form of tags which refer to a block-start/block-end pair enclosing the annotation in the annotation space, much as decimal label reference tags refer to a block-start/block-end pair surrounding text elsewhere in the main text space. Each new reference to the same text in the annotation space causes a new block-start/block-end pair to appear around the text. New references to existing annotation are made using the Refer To Annotation (RTA) command.

These annotation tags and blocks may also have keywords associated with them. As in keyworded jumps, these are a means of automatically following selected cross references in online display and Fullprint. The keyword string to be satisfied is set using the Set Keyword Annotation Request String (SKA) command. In online display, the selected blocks will appear immediately following the tags to those blocks separated with a display jump symbol (&) to signify the non-linear nature of the display. Annotation may contain any structure, including tags to other annotation blocks. All keyworded tags and jumps which are satisfied by the request strings will be followed. When the block end matching the block start referred to by a tag is encountered, another display jump symbol is shown and display returns to the point sequentially after the tag in the file.

Fullprint also follows keyworded annotation tags and jumps, but the selected blocks will appear as footnotes. Any nested annotation references will be ignored since one cannot have a footnote to a footnote printed out.

Annotation tags and blocks appear as other kinds of tags and blocks, followed by an or-bar (|) indicating their special purpose. Thus an annotation tag might appear as %T|"key" and an annotation block as %<|"key"...%>|.

The following is an example of a file containing some annotation tags and blocks.



2.7 PICTURES

When using FRESS on the Imlac, graphs, diagrams, and illustrations can be included with the text. These pictures are created or changed using the Sketch (SKE) command on the Imlac (see Imlac drawing package description). They are stored in the Picture Space of a file. References to a picture are made with the Make Picture Reference (MPR) command which creates a tag that refers to the picture. The picture itself can be displayed by jumping on the reference specified by the tag. If the user desires to reference a picture from another file, he must first copy it to that file with the Copy Picture (COP) command because picture references are not allowed to be interfile. A picture is deleted from a file with a special Delete Picture (DPI) command. This command will also delete all the tags referring to the picture. Other commands are: List Pictures (LP), which lists the names of all the pictures in a file; Change Picture Name (CPI), which changes the name of a picture; Scale Picture (SPI), which scales a picture; and Print Picture (PP), which draws the picture on a Calcomp plotter. The Sketch command and displaying a picture by jumping on a picture tag can only be done on the Imlac. All other sketchpad commands can be issued from any terminal.

3 VIEWING HYPERTEXT

3.1 KEYWORDS

Keywords can appear on blocks for retrieval purposes and on jumps for conditional splicing, as described in Sections 2.6.1 and 2.5.3. They are specified as optional parameters on the Make Block (MB), Insert Block (IBL), Make Decimal Block (MDB), Insert Decimal Block (IDB), Make Jump (MJ), Make Splice (MS), Make Annotation (MA), and Insert Annotation (IA) functions.

3.1.1 KEYWORD FORMAT

Keyword strings can be no longer than 255 characters in total and are made up of separate keywords or attribute-value pairs, separated by semicolons. Each keyword, attribute, or value is limited to 16 characters. No blanks are allowed in keywords. Any blanks specified will be ignored. Keywords are single identifiers. Values are like keywords but are used in conjunction with modifying attributes. For example, in a file consisting of popular song titles a user might wish to distinguish between those for which an author wrote the music and those for which he wrote the lyrics. The user could assign the attribute-value pairs:

```
composer:Irving_Berlin

lyricist:Irving_Berlin
```

Attributes are separated from their values by a colon. Only one value is allowed with an attribute and vice versa.

Keywords and values can also be weighted according to their importance to the section in which they are used. Weights range from 1-15 with 15 the most important. They must follow the keyword or value, separated by a comma (e.g., composer:Berlin,3 or Berlin,4). Only one weight is allowed per keyword or attribute-value pair. A weight of 0 or no weight at all indicates that the key is universal and will cause a match regardless of weight.

The following characters are not allowed in a keyword:

;	semi-colon	keyword separator
:	colon	attribute-value delimiter
,	comma	weight delimiter
&,│,¬,(,)	boolean operators	must be distinguishable for a retrieval request

Thus,

design;graphics:interactive;text processing,6

specifies a keyword string with keys "design" and "text processing", the latter with a weight of 6, and the attribute-value pair "graphics:interactive".

3.1.2 BOOLEAN RETRIEVAL

The rules for retrieving blocks (via the Block Trail (BT) command) and for matching keys on jumps (via the Set Keyword Jump Request String (SKJ) command) and annotation tags (via the Set Keyword Annotation Request String (SKA) command) are the same, as follows. With either type of request, a keyword in the request will match either a keyword or the value part of an attribute-value pair in the file; e.g., a request for the keyword "Irving_Berlin" would be satisfied by either attribute-value pair shown in Section 3.1.1. An attribute-value pair will match only another attribute-value pair. In this way the user can select only that music for which Berlin wrote the lyrics with no regard to the composer. If the "keys" (keywords or attribute-value pairs) match, then weights are compared. (See Section 3.1.1 for a description of weights.) If the request or the key in the file is unweighted the match is successful. If both are weighted then the key in the file must have a weight equal to or higher than the key in the request in order to satisfy the request. Boolean combinations of requested keys are satisfied using standard interpretations of the logical operators ¬(not), &(and), and │(or), in order of descending precedence. The precedence may be changed by the use of parentheses.

Example:

bt/keyword1&key2|key3
retrieves all blocks with both keyword1 and key2, as well as all blocks with key3

bt/keyword1&(key2|key3)
retrieves all blocks with both keyword1 and key2, and all blocks with both keyword1 and key3

bt/¬key1
is invalid since it requests all blocks which do not have key1 and it is not permitted to enumerate all keyworded blocks (possibly thousands).

3.2 SPACES

A file is divided into 9 logical units called spaces, of which 6 are of interest to the user. A few of these, in particular the main text and work spaces, have already been discussed in some detail. The System spaces are designed 1) to allow the online user to review what structure has been previously created, 2) to allow easy global editing (e.g., a spelling change on a keyword which is to be reflected throughout the entire file; this is called "inverse editing"), and 3) as an internal table for FRESS itself to obviate searching the main text space. Scrolling and pattern scanning may be done in these spaces.

3.2.1 MAIN TEXT SPACE

This space contains all the main text material in a file in one or more areas. Most of what has been discussed both in this manual and in the User's Guide has referred to the main text space.

3.2.2 WORK SPACE

A work space is provided for the FRESS user, in which he may collect sections of his text in one or more areas, edit them together, and then place the entire work space back into his main text at a desired location. He may also use the work space as an accumulator to collect text fragments for a separate document (useful for boilerplating applications) or to file them away in an "attic" where they will be gone from the main text but still accessible. To do this collecting, either Copy To Work (CTW) or Move To Work (MTW) may be used. To place the contents of the work space back in the main text, either Copy From Work (CFW) or Move From Work (MFW) may be used. These act on the entire work space.

3.2.3 ANNOTATION SPACE

The annotation space can contain text and all kinds of structure. However, it ordinarily holds only text which is annotation to material in the main text, work, or annotation spaces. A text string will be surrounded by a block-start/block-end pair for each reference to it. Lightpenning any of these block starts in a Jump (J) command will move the display to the corresponding tag, typically back in the main text space. For more information on the use of the annotation space, see Section 2.6.3.

3.2.4 KEYWORD SPACE

<u>The keyword space</u> is a system area storing the user's keywords alphabetically by keyword and attribute, and alphabetically by value for each attribute. Values are kept both with the attribute and as separate keywords. Along with each keyword or value is stored the number of pieces of each type of structure on which that word appears. A sample keyword space would appear like this:

```
design              3 %<
graphics
  design              3 %<
  hardware            1 %<
  interactive         2 %<
  passive             1 %<
  software            1 %<
graphics            1 %<
hardware            1 %<
interactive         2 %<
passive             1 %<
software            3 %<
sysproglang         1 %<  2 %J  2 %P
```

text processing	2 %<	2 %T

Note that the attribute "graphics," has 5 values in this file, and that the key "sysproglang" appears on 2 jump/pmuj pairs as well as on a block.

The keyword space is useful for making global changes on a particular keyword. If a keyword is editted in this space, the change will be reflected in all places in the file where that keyword is used (see Section 4.1.2). The keyword space can not, however, be used for travelling to any of the structure using these keywords.

3.2.5 LABEL SPACE

The label space contains all of the labels in a file in alphabetical order. They are displayed in columns, the number of columns being dependent on the current display width. Lightpenning a label in this space in a Jump (J) command is the same as doing a Get Label (GL) of that label.

3.2.6 STRUCTURE SPACE

The structure space of a file contains, in geographical order, all of the hypertext in a file. It reflects, in order, structure of the main text, annotation, and work spaces. Displaying the structure space can be a useful way to get an overall picture of the structure of a file, although it generally requires some familiarity with the hypertext functions of FRESS for one to effectively use it this way. Keywords, labels, and explainers are very helpful in increasing the amount of information to be gotten from the structure space. For instance, if keywords or labels are associated with the decimal blocks of a file, the structure space becomes a readable outline of the file.

A typical structure space of a file which contains 2 main text areas, a work space (with a label in it) and an annotation space might look like:

```
***AREA***
%< '1' (intro)
%< '1.1' (suba)%>
%< '1.2' (subb)%>%>
%Jjumps to 2nd area in text space%%
%< '2' (pro)%T"comment"%>
%< '3' (con)%>
***AREA***
%Pin 2nd text area%%
***AREA***
%<|"comment"%>|
***AREA***
%L(work)
*END OF STRUCTURE SPACE*
```

The work space has only one area line, a start line, for each area, and the annotation space has no area lines, so the reflection of the annotation space is 'sandwiched' between the end area line of the last text area and the start area line of the first area in the work space. In the above example, the annotation space contains only the block %<|"comment"%> and the work space contains only the label %L(work).

If any piece of structure in the structure space is lightpenned using the Jump (J) command, display will move to the structure in the main text, work, or annotation space which it reflects.

3.3 VIEWSPECS

Viewspecs (or viewing specifications) are a means of controlling which parts of a file will be visible in online display (or offline type), how much formatting is to be done, and whether certain dynamic features (such as keyworded jumps, viewspecs on jumps, etc.) are to be used. The command used to set the viewspecs in effect is Set Viewspecs (SV). There are four standard, system defined viewspec strings. These are:

edit	minimal online formatting, no right justification, format codes displayed
print	online formatting, no format codes displayed, no structure displayed, right justification
normal	online formatting, no justification, and format and structure codes displayed

* current viewspecs

Following one of these standard strings, the user may optionally include a list of specific viewspecs to be added or deleted from the standard string. Any number may be specified by giving the mnemonic codes preceded by a sign (+ or -) which indicates whether the viewspec is to be added or deleted. A given sign has scope over all viewspecs following it, until a new sign is encountered. Normally, the user specifies "*" plus or minus selected viewspec codes, giving him his current viewspecs with only a few changes. For example, had he just logged in to FRESS, issuing

```
sv /*-ofp
```

would display text formatted properly but without any format delimiters and codes.

The complete list of all possible viewspecs follows:

viewspec	explanation
=	points, block starts, block ends displayed
#	tags displayed
@	pmuj and pmuj explainers displayed
$	jump and jump explainers displayed
&	display jump symbol (&) displayed
AK	annotation keywords will be compared to string specified with SKANNOTATION
B	use special character for blanks
BL	display codes of form !nnn (encoding for special chars)
C	keywords displayed
CN	splices and ecilpses <u>not</u> displayed
D	decimal labels displayed
E	explainers displayed
FL	area lines <u>not</u> displayed
FN	block end numbers displayed
FP	format code delimiters (!) displayed
FV	formatting off
JK	jump keywords will be compared to string specified with SKJUMP
JU	text is justified
JV	jump viewspecs will be used
M	labels displayed
O	format codes displayed
P	viewspecs displayed
SP	structure code delimiters (%) displayed
S2	double space
T	regular text displayed

The viewspecs need not be specified as capital letters. The standard strings listed above can be expressed as the following combinations of viewspec options:

```
NORMAL is (#,=,@,$,P,FP,SP,T,E,O,D,C,M,JK,AK,FV)
EDIT is (NORMAL-FV)
PRINT is (T,JK,AK,JU,FV)
```

If the user would like to see what viewspecs are currently in effect, he uses the Display Viewspecs (DV) command.

4 HYPERTEXT EDITING

4.1 EDITING HYPERTEXT IN THE TEXT SPACE

Special rules apply to editing pieces of hypertext (blocks, jumps, labels, etc.) in the text space.

4.1.1 STRUCTURE IDENTIFICATION SYMBOLS

The first character after the % displayed for a piece of structure indicates what type it is. Specifying this character or the percent sign in an LP or scope indicates that the edit operation is to apply to the entire piece of structure. For instance, an insert would be placed after the last data field. Delete would remove the piece of structure with its data fields, and its other end if it were a block, jump, or annotation tag. (In the latter case, only the block start and block end, not the text of the annotation, are deleted.)

This rule also applies when any character in a *START OF TEXT AREA* or *END OF TEXT AREA* line is hit.

4.1.2 DATA FIELDS

Editing the data fields (see Section 2.2) of a piece of structure (label, keyword, or viewspec character strings) must produce a field which meets the restrictions imposed on the original specification. Except when editing labels, a Substitute in a data field actually does an insert of the new string and a delete of the old one, so the intermediate result after the insert must also be valid. For example, if the user wished to change the keyword 'sysproglang' to 'newest_lang', specifying

```
s/sysprog/newest_
```

would cause an error. The string 'newest_' would first be inserted, causing a keyword of length greater than 16. Instead, the following sequence of commands should be used:

```
de/sysprog
ib/lang/newest_
```

If the data field appears in a corresponding place in a control space, i.e., in the label, decimal label or keyword spaces, FRESS will automatically update the other occurrence. For the special "inverse editing" conventions which apply for editing in the various system spaces (label, keyword, etc.), see Section 4.2 on Editing in System Spaces.

4.1.3 MODIFIERS

Decimal blocks and splices are actually special cases of blocks and jumps respectively. The only difference in handling occurs in the case of Delete or Substitute. If the decimal number of a decimal block is specified in a Delete or Substitute, the special purpose is removed; that is, the block becomes non-decimal. The blank and apostrophe appearing on either side of the decimal number are considered part of the decimal number in this case.

Similarly, splices and ecilpses can be changed into jumps and pmujs. However, splices and ecilpses are displayed entirely differently from their associated general purpose structure (SP vs. J, EC vs. P). In this case, the <u>second</u> <u>letter</u> of the designation (P for SP, C for EC) serves the same purpose as a decimal number for a decimal block. If these symbols are indicated in a Delete or Substitute, the splice or ecilps will become a jump or pmuj. For example, if a splice appears in the file like this:

```
%SP"keyword"explainer%%
```

the command

```
de/P
```

would result in

```
%J"keyword"explainer%%
```

Note that there is no way to turn jumps into splices or non-decimal into decimal blocks.

4.1.4 LP'ING INAPPROPRIATE STRUCTURE

If a piece of structure or its data fields is LP'ed in a function which is inappropriate within hypertext, the structure is skipped when the LP is resolved. For instance, if the block

```
%<(label)"keyword"text in block%>
```

appeared in a file, a Make Jump (MJ) command specifying any part of the block, including the label and keyword data fields, as the LP would still result in

```
%<(label)"keyword"
%Jexplainer%%text in block%>
```

On the other hand, if a piece of hypertext is given as the first part of a Make Block (MB) scope, the piece of hypertext is included inside the block. This rule also applies to MOVE.

Occasionally the error message 'UNLIGHTPENNABLE TEXT FOUND BY PATTERN' will be given when an editing operation is attempted. This either means the user has attempted an edit when the viewspecs are set to right-justify the text, or the user attempted an illegal operation on hypertext, such as deleting the ***END OF LABELS*** line in the label space.

4.1.5 DELETION OF BLOCKS AND JUMPS

Deletion of a jump will of course delete its pmuj and all accompanying data fields. Similarly deleting a block start or end will cause the other end to be deleted. In addition, deleting a decimal block, or turning it into a non-decimal block, will delete all associated decimal label reference tags. Interfile jump and splice deletion is handled in exactly the same way except when the second file can not be found (because it was inadvertently scratched). In this case the message "WARNING: INTERFILE JUMPS FOUND WITHOUT PMUJ'S" will be printed, but the specified deletion will be completed. If the second file is passworded and not open, the message "PASSWORDED FILE NOT OPEN" will be printed, but, again, the specified deletion will be completed.

If a splice is included in the scope of an edit, the material affected will include the 'spliced-out' text. Thus if a file contained the string

```
This material will be deleted
```

with a splice added:

```
This material%SP&%ECdeleted
```

then specifying

```
de/This...deleted
```

will delete the whole string

```
This material will be deleted.
```

4.1.6 TEXT-ONLY EDITING

When the Delete (D), Substitute (S), Move (M), and Copy (CO) commands encounter hypertext during their execution, they will cause the message

```
HYPERTEXT: ACCEPT (A), REJECT(R), OR TEXT-ONLY(T)
```

to be printed. If the user wishes the command to be completed as specified (within the restriction that no hypertext can be moved interfile and no hypertext may be copied), he should type A. Similarly, he may reject the entire command by typing R. If he wishes the command to apply only to text (leaving the hypertext intact), he should type T.

If the user specified a piece of structure as his first LP in a Delete command, FRESS assumes that he wishes the command accepted and will not query the user during the execution of the command.

The user can specify that he wants the command to be accepted (thus supressing the message) by preceding the command name with a %, for instance, %D. It is not possible to specify text-only operations in the original command.

4.2 EDITING IN SYSTEM SPACES ("INVERSE EDITING")

4.2.1 LABEL SPACE

Simple editing (insert, delete, or substitute) is permitted in the label space and will be reflected as appropriate in the main text and decimal label spaces.

4.2.2 KEYWORD SPACE

Editing the text of a keyword in the keyword space causes the edit to be made on every occurrence of the keyword in the file. Only insert, delete, and substitute apply.

4.2.3 ANNOTATION SPACE

Deleting an annotation tag will delete the corresponding block start and end in the annotation space. If these were the last block start and end (i.e., there was only one reference to the block), a warning will be printed to notify the user. If the annotation block start and end are deleted, the associated tag will be deleted.

4.2.4 STRUCTURE SPACE

Editing of data fields in the structure space is permitted and will be appropriately reflected. It is also possible to rearrange (MOVE) the order of decimal label blocks and thus the text within them by manipulating them in the structure space. (See Section 5.3 for an example.) It is also possible to delete an entire block with text using the "block" qualifier in either text or system spaces (see Section 5.3). Certain types of editing are not allowed in the structure space. For instance, no regular text can be inserted since it is not clear where in the main text, work or annotation spaces it should be inserted. Similarly no new blocks can be made from within the structure space.

5 FRESS COMMANDS

5.1 FORM OF COMMANDS IN MANUAL

The section for each command begins with a designation of the format of the command. Parameters to be specified by the user take certain standard forms, explained in Section 5.2; they are enclosed in syntactic brackets ("<" and ">") in the command descriptions. The user must specify parameters in the order indicated, using the key delimiter of his choice (blank, /, etc.). For rules on specifying parameters in commands, see Section 3 of the User's Guide. Also, command mnemonics are shown below in upper case for ease of reading; they may be typed in upper or lower case, using any substring of the command name beginning with at least the underscored portion. Parameters followed by a raised o (°) are optional (see below). The symbol ␢ indicates a blank.

Parameters in square brackets ([,]) indicate parameters valid only in the multiple window or IMLAC version. They should be totally ignored when specifying commands in the single window version. Some comments also appear in square brackets, and these also refer to the multiple window version.

5.2 TYPES OF PARAMETERS

There are certain standard types of parameters in FRESS commands. These are listed below with an explanation of the format and meaning of each one.

5.2.1 <LP>

This parameter is a <u>L</u>ocation <u>P</u>ointer which designates a <u>position</u> in the file, often to indicate <u>where</u> some text or structure should be placed. It is specified as a context string of 1 or more characters which may contain an embedded ellipsis; the last character matched by the string is considered the <lp> except in the case of the Insert Before (IB) command or when & is ued (see Section 3.13 in the User's GUide). [<lp> may also be specified with the light pen on the IMLAC.]

5.2.2 <SCOPE>

This parameter designates an _amount_ of text to be affected by the command. It is specified either as a context string or as a pair of <lp>s identifying the beginning and ending points of the amount, one or both of which may be deferred.

5.2.3 <TEXT>

This parameter is a literal character string. It is a piece of text meant to be inserted in the file as specified in the particular command.

5.2.4 <LIT>

This parameter is a literal character string. Its use and format is determined by the particular command.

5.2.5 <N>

This parameter is a number. Unless otherwise indicated, it is an unsigned number to be considered positive. Its use is determined by the particular command.

5.2.6 <PASS>

This is a literal character string of no more than 7 characters which acts as a password to a file. Passwords are used to protect a file from particular commands, for instance to allow a user to scroll through but not edit a file. The password in effect is the one specified in the Get File (GF) command when the file was opened or in the most recent Change Password (CP) command. Any number of passwords may be associated with a file. If no password is specified when the file is created, the password DEFAULT is used and all commands are allowed.

5.2.7 <LABEL>

This is a literal character string of no more than 16 characters which acts as a label in a file. See Section 2.3 for further explanation of labels.

5.2.8 <BOOL>

This is a boolean request string, which is composed of keywords, attribute-value pairs, and weights combined with the logical operators ¬(not), &(and), and |(or), in order of descending precedence. The precedence may be changed by the use of parentheses. Examples of valid boolean request strings are:

```
key1&key2|key3
¬(key1&key2)&key3
```

For more information on boolean request strings, see Section 3.1.2.

5.2.9 <WIND>

This represents a window number. It must be an unsigned number. This type of parameter is used only in the multiple window version on the IMLAC.

5.2.10 <SPACE>

This parameter is specified as a single character, upper or lower case, representing one of six spaces (see Section 3.2). These are:

T	main text space
W	work space
L	label display space
K	keyword display space
A	annotation space
S	structure space

5.2.11 <FILE>

This is a filename, which is a literal character string of no more than 8 characters. If there is no <file> parameter in a command, the command is executed on the current file.

5.2.12 <VS>

This parameter is a viewspec string. Viewspecs are used to specify how the file is to be displayed and formatted online. See Section 3.3 for a complete description of viewspec strings.

5.2.13 <KEYS>

This parameter is a keyword string made up of keywords, attribute-value pairs, and weights. Each keyword must be no more than 16 characters long; the entire keyword string must be no more than 255 characters long. See Section 3.1 for a detailed explanation of keyword specification.

5.2.14 <PICT>

This is a picture name. It must be no more than 8 characters in length.

5.2.15 <OPTIONS>

This is a character string which may contain certain characters defined as indicating certain options available. The <options> list is unique to and defined in any command which uses it.

5.3 COMMAND QUALIFIERS

A "qualifier" is a keystroke saving way of specifying the amount of text to be affected by a command. A qualifier is specified by typing a hyphen (minus sign) and the qualifier code immediately after the command name, i.e., with no intervening blanks. Any command which has a <scope> parameter may be specified, instead, with a qualifier. When using the '-l' qualifier the <scope> parameter is then omitted from the command line. In all other cases, the <scope> is then specified by an <lp>.

The list of possible qualifiers is:

-c	character
-w	word
-l	line
-b	block

The <u>character</u> qualifier allows the user to specify an edit on a single character while still being able to give a longer text string to uniquely identify the character. For example, if the string

```
special blanks
```

appeared at the top of a display buffer and the user wished to delete the 's' at the end of the word 'blanks', he could not specify

```
d/s
```

since this would result in

```
pecial blanks
```

Instead, he can type

```
d-c/ks
```

which will delete the correct letter.

The _word_ qualifier allows the specification of any part of a word to indicate an edit on the whole word. Thus, to change the word 'sufficient' to 'adequate', the user need only type

```
     s-w/su/adequate
```

When used with the word qualifier, most of the commands containing <scope> parameters affect the blank before the word as well as the word itself. The exceptions to this are Capitalize, Make Block, Make Decimal Block, Make Annotation, Substitute, and Uncapitalize.

The _line_ qualifier specifies that the edit should apply to the first line of the online display and is especially useful when the text lines are cluttered with formatting and/or structure codes and it is difficult to lightpen exactly what is wanted.

The _block_ qualifier allows the user to specify an operation affecting an entire block (including the text inside the block) without having to LP both ends of the block. Thus

```
     m-b/%/<lp>
```

will move the specified block after the specified <lp>. Without the block qualifier, the user would probably have to defer the end of the scope parameter (what to move), move to the block end, and resolve the deferred scope.

The block qualifier can also be used to facilitate the rearrangement of blocks from within the structure space. For example, given a structure space which looks as follows:

```
     %< '1' (label)%>
     %< '2' (lab2)
     %< '2.1' (lab3)%>%>
```

To move the first block (with decimal label 1) to after the last block (terminated by %>%>), the user would specify:

```
     m-b/1/%>>%>>
```

Because of the block qualifier, the user can specify the decimal number attached to the block instead of the % of the block to be moved.

When manipulating blocks in the structure space using a standard terminal (with no lightpen), the user will often have difficult LP'ing the correct block end. The problem can be solved by deferring the <lp>, then scrolling forward or pattern scanning until the desired block end appears in the first display line. It is then much easier to specify the correct <lp>.

5.4 SPECIFYING PARAMETERS

Some functions allow the user to omit values for some parameters. These parameters are called optional and are indicated in the command descriptions by a degree sign (°). There are a few forms which commands may take, and these are described below with examples of the assignment of parameters.

No matter what form a command takes, however, it is logically impossible for the user to specify too many input values (except for functions which take no input values). After all required and optional parameters have been assigned, the remaining text on the command line is taken as part of the last parameter. For example, the Insert (I) command takes only 2 parameters. If the user typed

```
I/LP/text/more text
```

then the <text> string to be inserted would be

```
text/more text
```

It _is_ possible to specify too few input values. If the number of values specified is less than the number of required parameters, an error message is printed.

All parameter assignment is done in the order in which the input values are specified on the command line. Also, all required parameters are assigned values before any optional parameters are assigned values. In order to minimize keystroking, shortcuts exist for the experienced user; novices may avoid having to learn these methods by sticking to the straightforward consistent specification method presented first.

5.4.1 THE SIMPLE WAY

The simplest way to specify a command line is to use key delimiters for <u>all</u> parameters, including those which are to be left null. As an example, consider the Insert Decimal Block command, which is of the form

```
IDB <lp>° <label>° <keys>° <text>
```

If the user wished to insert a decimal block after the word 'here', wished it to contain the words 'inside the block', and wanted no keywords or label on the block, he would type

```
IDB/here///inside the block
```

specifically leaving out the label and keyword fields. This method can be somewhat awkward, for instance in using the Make Jump command, which is of the form:

```
MJ <lp1> <lp2> <text1>° <text2>° <vs1>° <vs2>° <keys>°
```

If the user wished to specify values for <text1>, the explainer on the jump, but leave out all other optional parameters, this method would have him type:

```
MJ/word1/word2/expl1////
```

To alleviate this problem, one additional rule can be applied. If, as in the example above, all the remaining parameters not yet specified are optional, and the user does not wish to assign values to any of these, he may omit all of them. Thus, instead of the awkward example above, he would type:

```
MJ/word1/word2/expl1
```

5.4.2 A MORE COMPLEX WAY

The same rules apply to all parameter assignments. However, it is easiest to explain these rules if commands are classified according to the type and order of their parameters.

5.4.2.1 All parameters required

If all parameters of a command are required, then the user must specify values for each of these parameters, and they are assigned in order. For example, the Copy File command is of the form

```
     CF <file1> <file2>
```

where <file1> is the file being copied and <file2> is the file being copied to. All parameters are required, so any specification of this command will look similar:

```
     CF/intro/intro2
```

5.4.2.2 All parameters optional

If all parameters of a command are optional, the user may specify any number of them. They are assigned in the order they are given, from left to right. If he wishes to specify a value for a parameter which appears in the command description after a parameter for which he does not wish to specify a value, the latter must be specifically left null, that is, a key delimiter for it must appear in the command line. For example, the Fullprint command is of the form

```
     FU <options>° <file>° <pass>° <lp>°
```

If the user wished to specify a filename, but no <options>, he would type

```
     FU//filename
```

Since assignment of the parameters is done left to right, no key delimiters need be specified for the optional parameters after the <file> in the command description if no values are to be assigned to them. Thus if the user did not wish to specify values for any parameters, he would type simply

```
     FU
```

5.4.2.3 Mixture of optional and required

When both optional and required parameters appear in the same command, all of the optional parameters are grouped together. Required parameters, on the other hand, may be in two groups separated by optional parameters in the middle. Currently, however, no commands exist which have this format; commands have either leading (required followed by optional) or trailing (optional followed by required) required parameters, but not both.

Values are assigned first to all required parameters. If the required parameters are before the optional ones, then all parameter assignment is done left to right. For example, the Make Jump command has two required parameters followed by five optional ones:

```
     MJ <lp1> <lp2> <text1>° <text2>° <vs1>° <vs2>°
<keys>°
```

If the user typed the command line

```
     MJ/here/there/expl1//print
```

this would result in the following assignment of values:

```
     lp1   = here
     lp2   = there
     text1 = expl1
     vs1   = print
```

Because of the left-to-right assignment, <text2> must be specifically left null in order to assign a value to <vs1>.

If the required parameters are after the optional ones, the last values specified on the command line are assigned to the trailing required parameters. Then the remaining values are assigned, left to right, to the optional parameters. Consider the Insert Block command, which is of the form

```
     IBL <lp>° <label>° <keys>° <text>
```

If the user specified the command line

```
     IBL/here/words inside block
```

the value 'words inside block' would be assigned to the required parameter <text>, then the remaining value, 'here', would be assigned to the first optional parameter <lp>.

5.4.3 IMPLIED INSERT POINTS

Many commands have an optional <lp> parameter. Except for the Fullprint command, if no value is specifically given to this parameter by the user, the implied insert point is used. For example, the Make Decimal Reference command is of the form

```
MDR <lp>° <n>
```

where <n> is the decimal number of the block to be referenced. If the user specified

```
MDR/2.1
```

a decimal label reference tag referring to block 2.1 would be created at the implied insert point.

5.4.4 NULL REQUIRED <LP>S AND <SCOPE>S

Required parameters may not be entirely omitted, as optional parameters may. However, any required <lp> or <scope> parameter may be left specifically _null_, by including a key delimiter but no explicit value for it. In this case, the first character of the display buffer is used as the default. For example, consider the Capitalize command, which is of the form

```
CA <scope>
```

The user may _not_ simply type

```
CA
```

which would be omitting the required parameter. He _may_, however, type

```
CA/
```

which includes a key delimiter for the parameter and thus specifically nulls it. In this case, the first character of the display buffer would be capitalized. As another example, consider the Copy command, which is of the form

```
CO <scope> <lp>
```

Either one (or both) of these parameters may be left null. The user might specify

```
CO/pattern/
```

which would copy the word 'pattern' after the first character of the display buffer, or

```
CO//there
```

which would copy the first character of the buffer after the word 'there'. Specifying

```
CO//
```

would cause the first character of the buffer to be copied after itself.

5.4.5 INPUT MODE

Besides the Make File (MF), Bottom Input (BI) and Top Input (TI) commands, which specifically enter Input Mode, certain other commands may be used to enter Input Mode. If the <text> parameter is left null in the Substitute (S), Insert (I), Insert Decimal Block (IDB), Insert Block (IBL), or Insert Annotation (IA) commands, the user will be placed in Input Mode at the point at which the <text> would be inserted if it had been included explicitly in the command. For example, the Substitute command is of the form

```
S <scope> <text>
```

If the file contained the string 'The word was' and the user specified

```
S/word/
```

he would be placed in Input Mode after 'The '. The Insert Annotation command is of the form

```
IA <lp>° <keys>° <text>
```

If the user specified

```
IA/here/keyword/
```

the user would be placed in Input Mode inside the newly created annotation block.

When the user leaves Input Mode, the display point is one word before the last line inputted.

5.5 ALPHABETICAL LISTING OF FRESS COMMANDS

A

• Accept

```
ACCEPT
```

Normally, FRESS makes permanent the result of the previous editing operation each time the current edit is executed. Accept, however, explicitly makes permanent the current editing operation, writing it in the user's file on disk, and thereby disabling the Revert function. This command should be used if a user has to leave his terminal for an extended period of time so that the last edit will not be lost if the computer should drop him.

• Add Password

```
APASSWORD <pass> <lit>
```

<pass> is the password to be added
<lit> is the 'allowable functions list' (see below)

Each password assigned to a file has associated with it an 'allowable functions list', that is, the list of FRESS functions (commands) which the usser is permitted to execute when viewing the file using that password. When a file is created with the Make File (MF) command, the password assigned to the file (DEFAULT if none is specified) has an 'allowable functions list' which includes all FRESS commands. Later on, the owner of the file may wish to allow others to use the file (for editing, viewing, etc.) but under protected conditions. For example, he may wish a certain user to be able to display and scroll through all portions of the text but not be able to make any changes in the text.

To accomplish this, the user adds new passwords to his file (a password may be up to seven characters long and contain any characters), designating for each password the functions he desires the particular user who accesses the file with that password to be able to perform.

The following format for the <allowable functions list> is used:

<standard string> <sign> <function> ... <function> <sign> <function> ... function>

Blanks must be used to separate the <functions>'s not separated by <sign>'s. Each <function> is indicated by any substring which would be considered legal when actually specifying the command.

System defined <standard strings> are specified by:

ALL	all functions allowed
NONE	no functions allowed
DISPLAY	only display functions
*	same functions as present password

The <sign> mentioned above specifies whether the function is to be permitted (+) or prohibited (-). A given sign has scope over all function mnemonics encountered before the next sign. Thus, if the user opened a file with a password allowing all functions, he could create a password (in the example below, "pass") which does not allow insert and delete as follows:

```
ap /pass/*-i d
```

It is possible to add the system password "DEFAULT" to one's file, allowing it to be accessed in the future without password specification. Specify:

```
AP default <allowable functions>
```

B

• Bottom of Space

```
BOTTOM <space>° [<wind>°]
```

This command moves the display pointer to the last display line of the last area of the space indicated by <space> [in the file in the window specified by <window#>], and displays that line [in window <window#>]. If <space> is not specified, the display will move to the bottom of the space currently being viewed. [If no window number is specified, the current window is assumed.]

• Bars

```
BARS <scope>
```

This command causes the <scope> to be surrounded by !-r- format codes. When Fullprint (FU) is used to print the file, revision bars will appear to the left of the left margin on the lines containing the text indicated in <scope>. This facility offers a good way to show selective changes in large documents.

• Blank Window (multiple window version)

```
BWINDOW <wind>°
```

This command will remove from the screen all text in the specified window. If no window is specified, the current window is blanked. A non-linear travel function (e.g., DSPACE, GLABEL, etc.) must be used to return text to the window.

This command is useful for blanking a window which would otherwise be regenerated (redisplayed) when an edit is done in another window. Where possible, only changed windows are regenerated, but windows containing pictures and right-justified text are always regenerated.

• Bottom Input

```
BINPUT <space>°
```

This command causes input mode to be entered at the bottom of the last area in the specified space. Valid spaces are text and work. If no space is specified, the current space is assumed.

• Block Trail

```
BTCONTINUOUS <bool>° [<wind>°]
BTDISCRETE <bool>° [<wind>°]
```

The block trail command retrieves all the keyworded blocks in the file whose keywords satisfy the boolean request (see Section 3.1.2) and displays at the top of the first one. [In the multiple window version, the block is displayed in the specified window. If none is specified, some window other than the current one is used.] With the continuous trail, all the blocks are displayed contiguously, one after the other. Thus, scrolling over the end of a block in a continuous trail will automatically cause the display to move to the top of the next block in the trail. The TRAIL function may also be used. The user may edit, scroll, and pattern scan through a trail. The block trail environment may be left by any non-linear travel function (e.g., Display Space, Get Label).

A discrete trail contains the same blocks as a continuous one, but it causes the blocks to be displayed singly in their normal context in the file instead of in a chain. The user must specifically request to go the next (or previous) block using the Trail command.

Example:

```
bt/keyword1&key2|key3
```

retrieves all blocks with both keyword1 and key2, as well as all blocks with key3

bt/keyword1&(key2|key3)
retrieves all blocks with both keyword1 and key2, and all blocks with both keyword1 and key3

bt/¬key1
is invalid since it requests all blocks which do not have key1 and it is not permitted to enumerate all keyworded blocks (possibly thousands).

Error messages occur in the case of the last example, or when a keyword in the request is not referenced in the file, or when the form of the request is invalid (mismatched parentheses, two &'s in a row, etc.)

If weights are specified in the boolean request, the blocks are retrieved in order of weight, from highest to lowest. For instance:

bt key,6	retrieves all blocks having key with a weight between 15 and 6, or no weight (which satisfies all requests).

C

• Capitalize Text

```
CAPITALIZE <scope>
```

<scope> is the string to be capitalized

This command capitalizes the indicated text. All regular text or jump explainers can be capitalized. As an example, the following display line.

    All regular text or jump explainers

and the command line:

    CAP reg...ners

would result in:

    All REGULAR TEXT OR JUMP EXPLAINERS

• Copy File

```
CFILE <file1> <file2>
```

<file1> is the file to be copied
<file2> is the file to be copied into

CFILE allows the copying of one file into another. The message "COPY SUCCESSFUL" is typed upon successful completion. Error messages are generated if <file2> already exists, if <file1> does not exist, or if an invalid filename is typed. Under CMS this command is used strictly for making backup copies, saving intermediate drafts or generating multiple versions of a document.

• Copy From Work Space

$\boxed{\text{\underline{CFW}ORK <lp>}^{o}}$

<lp> is the point to copy after; if omitted, the implied insert point is used

This command copies all the text currently in the work space (see Copy To Work and Move To Work) into the main text after the point specified. The text also remains unaltered in the work space. The text from <u>all</u> areas in the work space is copied - no selectivity is possible. However, the area lines themselves can not be copied, so a text-only edit (see Section 4.1.6) should be done. If the user does not wish the whole space to be copied, an ordinary COPY should be used, with the <lp> deferred.

• Copy Text

$\boxed{\text{\underline{CO}PY <scope> <lp>}}$

<scope> is the amount of text to be copied
<lp> is the point to copy after

The string of text specified by the first parameter is copied after the point indicated by the last parameter. The text remains unchanged in its original location.

Applied to the previous sentence, for example, the command

```
co /unchangedb/ginalb
```

would result in

"The text remains unchanged in its original unchanged location"

Format codes may be included in the text being copied; however, to preserve uniqueness, structure codes (e.g., labels, jumps and pmujs) in the original string may not be copied.

To copy text from one file or space to another, the user may defer the "to" point, travel to another file or space using either Get Label, or Get File, and then resolve the last point. For example, to copy the previous paragraph to the start of an existing file (just after the start line, "*START OF TEXT AREA*"), the following sequence could be used:

```
          co /Format...copied./?
          gf existing
          ? /*
```

• Copy Picture

```
COPICT <pict> <file1> <file2>
```

<pict> is the picture to be copied
<file1> is the file the picture is to be copied from
<file2> is the file the picture is to be copied to

This command copies the picture specified by <pict> from <file1> to <file2>. <file1> is unchanged. If the picture <pict> already exists in <file2>, the copy will not be done.

• Change Password

```
CPASSWORD <pass>
```

Once the user has accessed a file, he may switch to a different password under which he wishes to operate on the file, that is, he may change his 'present' password. For example, if he has accessed the file with a password which permits all functions and wishes to let someone view his file without being able to perform any alterations, he may switch to a restricted-function password without the bother of freeing the file and getting it again with the different password. (This lesser password would hopefully not allow a Change Password function!) Note that this function merely switches to a password previously created (using AP); it does not create a new password.

• Change Picture Name

```
CPICT <pict1> <pict2>
```

<pict1> is the old name of the picture
<pict2> is the new name of the picture

This command will change the name of a picture from <pict1> to <pict2>. The name will be changed in all of the references (tags made with the Make Picture Reference command) to the picture as well.

• Copy to Label

```
CTLABEL <scope> <label>
```

<scope> is the amount to be copied
<label> is the label after which to copy the text

CTLABEL copies the specified text to the point immediately after the specified label (see MLABEL). As with all copy operations, the text also remains in its original location. To gather miscellaneous notes on a single subject, for example, the user might create an appropriate label, then scan through his text copying relevant sections to this label. This command cannot be used to do interfile copies.

• Copy to Work Space

```
CTWORK <scope>
```

<scope> is the amount to be copied

When the user specifies a CTWORK command, the desired text will be copied to the bottom of the last area in the work space, and will also remain in the main text body.

For the difference between COPY and MOVE, consult those functions themselves.

• Change Current Window (multiple window version)

```
CWINDOW <wind>
```

This command makes the specified window the current window. All functions which default to the current window if no window is specified will then use this window.

• Delete Text

$\boxed{\text{\underline{D}ELETE <scope>}}$

<scope> is the amount to be deleted

The FRESS Delete command allows the user to delete an arbitrary portion of his FRESS file. Given the string

SYSTEM SHUTDOWN AT 2200 FOR TWO HOURS..

the command line

de TWOø

would result in the string

SYSTEM SHUTDOWN AT 2200 FOR HOURS..

The use of the ellipses ("Dot Dot Dot") specification simplifies the deletion of longer strings of text. For example,

de / SH...WO

would result in

SYSTEM HOURS..

Delete is an example of a function for which qualifiers are useful; for example, if the top of the display buffer were positioned at the start of this paragraph, "de-w us" would delete "useful"; "de-c unct" would delete the "t" in "function". See Section 5.3 for further information on qualifiers.

For a discussion of Hypertext deletion see section 4.1, Editing Hypertext in the Text Space.

• Delete Password

$\boxed{\text{\underline{DP}ASSWORD <pass>}}$

The delete password command deletes the specified password from the current file. The default password, "default" can be deleted using this command. Care must be taken not to delete all passwords in one's file, after which the file cannot be opened.

• Delete Picture

$\boxed{\text{\underline{DPI}CT <pict>}}$

This command will delete the picture specified by <pict> and all references (tags) to it.

• Display Space

$\boxed{\text{\underline{DS}PACE <space>° [<wind>°]}}$

Display Space is a quick way of traveling to the start of certain system-defined spaces in a FRESS file. The spaces which are currently available (see Section 3.2 for a description of each):

<space>	(description)
T	main text space
W	work space
L	label display space
K	keyword display space
A	annotation space
S	structure space

If no <space> is specified, the current space is assumed. [In the multiple window version, the space is chosen from the file in the specified window, and is displayed in the specified window. If none is specified, the current window is assumed.]

• Display Viewspecs

$\boxed{\underline{\text{DV}}\text{IEW}}$

This command allows the user to see what the current "viewspecs" are (see Section 3.3). The viewspecs determine what the user will and will not see when displaying his file. For example, he can turn off the Online Formatting capability of the system, or prevent the display of formatting or structure codes.

The viewspecs are expressed in terms of one of 4 standard viewspec strings and additions or deletions to these strings. These four standard strings correspond to the standard strings given in the Set Viewspecs command:

VS1	normal
VS2	print
VS3	edit
VS4	null string

Some examples are:

```
VS1+FL
VS3+CN B-JK M
```

vs4 is used when none of the other standard strings are close to what the viewspecs actually are. Thus

```
VS4+T FL
```

means only T and FL viewspec options are in effect.

<u>E</u>

• End FRESS Session

$\boxed{\underline{\text{E}}\text{ND}}$

The END command closes all open FRESS files (equivalent to free-filing (see FFILE below) <u>all</u> files being used) and ends the FRESS editing session. The user is returned to the CMS command environment.

F

• Free File

```
FFILE <file>
```

This command makes permanent the last editing operation specified and "frees" the specified file. Note that ACCEPT makes the change permanent also but does not close the file.

• Make Footnote

```
FOOTNOTE <scope> <n>°
```

<scope> is the text to be made into a footnote
<n> is the optional footnote number

This command causes the specified text to become a footnote by inserting footnote format codes, (!-f-) around the text. If a footnote number (n) is specified, !-fn- will be inserted in front of the text. For example, if the display were located at the start of this paragraph:

```
fo/command...become/3
```

would result in:

This !-f3-command causes the specified text to become!-f- a footnote.

• Full Printout

```
FULLPRINT <options>° <file>° <pass>° <lp>°
```

This command is used for paginated and fully formatted hard copy output. If no parameters are specified the current open file is printed to the offline printer.

<options> = <option,option,...,option>

AT<n>	the first page printed is numbered <n>. If AT<n> is not specified the default value is FROM<n>. AT0 suppresses all page numbering for that printout.
FROM<n>	the first page printed on the output device is page <n> i.e., output from previously formatted pages is suppressed. <n> is the nth physical page, not the nth numbered page.
HYPER	format codes are printed along with the text and are also interpreted. This command is useful when one wants to debug one's format codes, but overlaying may occur with tabs (see section 5.11 of the User's Guide).
MACROS	macros (inserted by user in the file) are used to determine the format of each decimal level (number of spaces indented, heading format.) See section 2.6.2 for further explanation.
OFFSET<n>	shifts the output <n> spaces to the right on the output device. Note that this offset is performed on all pages and is not related to the OFFSET ALTER code. <n> plus the value of the WIDTH ALTER code should not exceed 110.
PDISK	creates a file of filetype "print" on the user's CP/CMS P-DISK. The filename is the same as that of the FRESS file printed. The output file must be printed on the offline printer using the CMS "offline printcc" command.
STOP	waits for a carriage return before printing the next page. If STOP is specified, the user's output is automatically routed to his terminal. This option is useful for printing on non-continuous forms, e.g., letterhead on a typewriter-like terminal.
TO<n>	the last page printed is <n>, i.e., the total number of pages printed is TO<n>-FROM<n>+1
UPPER	translates to upper case
V	translates special characters so that they print on a terminal as an analogous character (i.e., "[" prints as "(") rather than as a blank when using the 2741 option (below).

2741	print to user's terminal. When this option is specified the user must wait for a 'proceed' signal, meaning the FULLPRINT program is ready to start printing. The terminal paper should be adjusted if necessary. The user must hit carriage return before the FULLPRINT will start.
1403	print to 1403 (offline) printer

Defaults:

```
fu /FROM1,TO32000,AT1,1403,OFFSET0
```

For minimally formatted hard copy output which includes both format codes and structure, use Offline Type. For further information, see the explanation of that command.

<file> <pass>

The user may override printing of the current open file by specifying a filename and, if the file is password protected, a password.

<lp>

The user may begin printing at any location in his file by specifying a lightpen hit. Page numbering will begin at 1. Because the FULLPRINT program runs independently of any online formatting, the user must be sure that FULLPRINT knows about any macro definitions he has embedded in the file (see Section 5.7 of the User's Guide). The optional <lp> parameter is not defaulted to the implied insert point.

G

• Get File

```
GFILE <file> [<wind>°] <pass>°
```

This command is used for two purposes. First, it is used to gain initial access to an existing file after entering the FRESS environment. (If creating a new file, a user will issue the MFILE command.) FRESS will open the specified file, and the display buffer is set at the "*START OF TEXT AREA*" line, but no text is printed. [In the multiple window version, the buffer is displayed.] The user may then utilize any FRESS function within the capabilities of his password. No <pass> need be specified if the file has the system default password ("DEFAULT"). There is currently no limit on the number of open files other than the availability of machine 'core'. This may be altered in other versions.

Once a file has been retrieved using the Get File or Make File commands, or by scrolling or jumping into it from another file, it is considered open until it is specified in a Free File command. While the file is open, however, Get File may be used for the second purpose of making a file the current file. For instance, if the user opened three files consecutively and then wished to return to the first of these files to do some editing, he would use the Get File command for this purpose.

[In the multiple window version, the file is displayed in the specified window. If none is specified, the current window is assumed.]

• Get Decimal Label

```
GDLABEL <n> [<wind>°] <file>°
```

<n> is the number of the decimal block to be retrieved

The GDLABEL command is similar to the GLABEL command (see below) except that it locates the block which currently bears the decimal label specified (See Decimally Labelled Blocks, Section

2.6.2). If a <file> is specified, that file is searched for the block.

[If a window number is specified, the searching is done in the file displayed in that window (unless <file> is specified), and the block will be displayed in that window. If no window is specified, the current window is assumed.]

• Get Label

$\boxed{\underline{\text{GL}}\text{ABEL <label> [<wind>}^{\circ}\text{] <file>}^{\circ}}$

To travel quickly and conveniently from place to place in a FRESS file, one can label various points (using MLABEL; see below). Get Label causes FRESS to locate the point in the file at which the label is defined and to start the display at that point. A substring of the label may be specified, and the (alphabetically) first label beginning with that substring will be retrieved.

The second parameter (<file>) is normally omitted, which means that only the FRESS file currently being displayed is searched for the label. However, if a filename is specified, that file (which must be open at the time) is searched for the label. If an asterisk (*) is given for the filename, then all open files are searched for the label. Thus GLABEL provides a convenient means of interfile traveling.

[If a window number is specified, the search for the label will be done in the file displayed in that window (unless <file> is specified) and the label will be displayed in that window. If none is specified, the current window is assumed.]

<u>I</u>

<u>• Insert Text</u>

```
INSERT <lp>° <text>
```

<lp> is the point to insert after; if omitted, the implied insert point is used
<text> is the character string to be inserted; if left null, Input Mode is entered

The Insert command allows the insertion of an arbitrary length string of text after a specified location in the user's file. Given the fragment

```
See Spot run
```

the command

```
i/run/.
```

will create the sentence

```
See Spot run.
```

Of course, longer inserts are possible. One might specify:

```
i /Spot/, Dick and Jane's pooch,
```

yielding

```
See Spot, Dick and Jane's pooch, run.
```

For multiple line inserts, the user may omit the <text> parameter, which will place him in Input Mode after the indicated <lp>. If no <lp> is specified, the implied insert point is used.

• Insert Annotation

```
IANNOTATION <lp>° <keys>° <text>
```

<lp> is the place for the tag to go; if omitted, the implied insert point is used
<keys> are the keywords to be placed on the annotation tag and block
<text> is the character string to be inserted in the annotation block; if left null, Input Mode is entered inside the annotation block

The specified text is placed in an annotation block in the annotation space, and a tag referencing the block is placed in the indicated place in the text. If no text is specified, input mode will be entered in the annotation block. After the command, display will be in the annotation space. To return to the tag, do a "return". The new implied insert point will be after the text inserted in the annotation block.

See the SKA command for information on the displayed format of annotations.

• Insert Before

```
IBEFORE <lp> <text>
```

<lp> is the place to insert the text before
<text> is the string to be inserted

The Insert Before command is similar to the Insert command, except that it inserts the text before the specified <lp> instead of after it. In this case, the <lp> is considered the first character specified in the context string. Insert Before does not use the implied insert point as does Insert, nor does it set the implied insert point. The user may not enter Input Mode using Insert Before. Insert Before can be used only in the main text, work, and annotation spaces.

• Insert Block

$\boxed{\text{\underline{IBL}OCK <lp>}^{o}\text{ <label>}^{o}\text{ <keys>}^{o}\text{ <text>}}$

<lp> is the place to make the block after; if omitted, the implied insert point is used
<text> is the string to be inserted in the block; if left null, Input Mode is entered

IBLOCK creates a new block at the specified LP. The text given in the last parameter is inserted inside the new block. If the last parameter is left null, Input Mode is entered positioned inside the block. Thus

```
IBL/here/lab//
```

indicates that an unkeyworded but labeled block at 'here' is to be entered in Input Mode. Also,

```
IBL/text string
```

puts the text string at the implied insert pointer, without labels or keywords.

The result of IBLOCK and MBLOCK are the same, except that the MBLOCK command requires that the text inside the block has been previously entered in the file.

For more examples using this command see Section 2.6.2.5.

• Insert Decimal Block

```
IDBLOCK <lp>° <label>° <keys>° <text>
```

<lp> is the point to make the block after; if omitted, the implied insert point is used
<text> is the string to be inserted in the block; if left null, Input Mode is entered

Insert Decimal Block is similar to the IBLOCK command except that it makes a decimal label block. Note that IDBLOCK is the typical means used to input the sections of a decimally structured file in Input Mode. See Section 2.6.2 for more detailed discussion of decimal blocks.

J

• Jump

```
JUMP <lp> [<wind>°]
```

Jump is a very powerful command for travelling non-linearly in a file. The <lp> can be in the main, annotation, or work spaces (referred to as main spaces), or the structure or label spaces. If the <lp> is in the label space, the effect is a GLABEL of the specified label. If the <lp> is in the structure space, display will be at the corresponding order in a main space. Jump's function is somewhat different if the <lp> is in a main space, as is indicated by the following table:

MAIN SPACE LP	NEXT DISPLAY STARTS AT:
label	that label in structure space
point, area order	corresponding structure space location
block start(end)	matched end (start) unless <lp> is of a block start in the annotation space, in which case the next display will be at the tag which references the block.
jump(pmuj)	matched pmuj(jump)
tag (annotation, decimal label reference, picture reference)	place referenced by tag
keyword of order	same as <lp> of order
explainer	same as jump or pmuj
other text	equivalent to a locate, except the <lp> character is considered to be the last character rather than the first character if a literal string is specified.

In all cases (in the single window version) the old display point will be saved and can be returned to by specifying "return". [In the multiple window version, the "jumped to" point is displayed in the specified window. If none is specified, some window other than the current one is assumed.]

<u>L</u>

• <u>Locate (Pattern Scan)</u>

```
LOCATE <lit>° [<wind>°]
```

<lit> is the literal character string to be located

Other LOCATE commands may be created using the modifiers <u>L</u>ong, <u>B</u>ackwards, and <u>M</u>ixed, in any order:

<u>LB</u>	<u>L</u>ocate <u>B</u>ackwards
<u>LL</u>	<u>L</u>ocate <u>L</u>ong (to bottom of area)
<u>LM</u>	<u>L</u>ocate in <u>M</u>ixed mode (take specified character case literally, without folding)
<u>LBL</u>,<u>LLB</u>	<u>L</u>ocate <u>B</u>ackwards <u>L</u>ong (to top of area)
<u>LBM</u>,<u>LMB</u>	<u>L</u>ocate <u>B</u>ackwards in <u>M</u>ixed mode
<u>LLM</u>,<u>LML</u>	<u>L</u>ocate <u>L</u>ong to bottom of area in <u>M</u>ixed mode
<u>LBLM</u>,<u>LBML</u>,<u>LLBM</u>,<u>LLMB</u>,<u>LMBL</u>,<u>LMLB</u>	<u>L</u>ocate <u>B</u>ackwards to top of area in Mixed mode

The pattern scanning facilities of FRESS allow the user to search a FRESS file for the first occurrence of the pattern specified and to begin the display at the start of the matched string. If the simple form of locate is specified without a text parameter, i.e., just typing 'l', the previous locate command, regardless of form, is repeated. If any other form of locate is specified without a <lit> parameter, e.g. just typing 'lb' or 'lm', the pattern specified in the previous locate command (of any form) will be used. If no previous locate was done, an error message is printed. As with the specification of Location Points and amounts of text (Scopes), the pattern may contain the ellipses (...) construct, but this will take considerably more execution time!

By use of the various forms of the locate command, the user is able to control how the pattern is searched for. For LOCATE, the pattern is matched without regard to upper and lower case ("folded"), and the scan is in the forward direction for up to 8000 characters. If a "B" is used in the command name, the scan is done in a backwards direction for 8000 characters. If a second "L" is used in the command name, the scan will continue until the end of the area (top or bottom) regardless of length. (This scan will also continue across splices, so care must be taken when it is specified.) If an "M" is used in the command name, the pattern is scanned for exactly as typed (in "mixed mode"), i.e. without folding. [In the multiple window version, the pattern scan is done in the specified window. If none is specified, the current window is assumed.]

Failure to locate a pattern will result in one of the following messages:

1. "PATTERN NOT FOUND, EO COUNT" - the specified pattern was not found in 8000 characters from the current display point. This error will not occur when the "L" modifier is used.
2. "PATTERN NOT FOUND, EO AREA" - the specified pattern was not found within the limit of the area in the direction specified.

• List Pictures

```
LPICT
```

This command will list the names of all the pictures in the current file.

<u>M</u>

<u>• Move Text</u>

$\boxed{\text{\underline{M}OVE <scope> <lp>}}$

<scope> is the amount of text to be moved
<lp> is the place to move the text to

This command behaves much like COPY, except that the specified text is deleted from its original location. A MOVE of a large block of text is often done using DEFERRED LP's, discussed in Section 3.6 of the User's Guide. Considering a small example, modifying:

This is a DATEL terminal.

with the command

```
mo / DATEL/al
```

will result in

This is a terminal DATEL.

No hypertext can be moved interfile.

<u>• Make Annotation</u>

$\boxed{\text{\underline{MA}NNOTATION <scope> <keys>°}}$

<scope> is the text to be made into an annotation

The specified text is placed in a block in the annotation space, and a tag referencing the block replaces the text itself in the text or work space. Any keywords specified will be placed on the block and the tag. The display after the command is executed starts where the tag was inserted.

• Make Block

$\boxed{\text{\underline{MB}LOCK <scope> <label>}^{o}\text{ <keys>}^{o}}$

<scope> is the text to be included in the block

This command creates a block containing the specified text. The text must already exist in the file. Empty blocks can not be made. If one character is specified, the block is made around that character.

Thus,

```
MB/The d...em/labx/design;graphics:interactive;text processing,6
```

specifies that the sentence "The d...em." is to be made into a keyworded block with keys "design" and "text processing", the latter with a weight of 6, and the attribute-value pair "graphics:interactive", and with label "labx".

Two implied insert pointers are created by this command, as explained in Section 2.6.2.5.

• Make Decimal Block

$\boxed{\text{\underline{MD}BLOCK <scope> <label>}^{o}\text{ <keys>}^{o}}$

<scope> is the text to be included in the block

MDBLOCK is similar to the MBLOCK command except that it makes a decimal label block (see Decimally Labelled Blocks, Section 2.6.2).

• Make Decimal Reference

$\boxed{\text{\underline{MDR}EF <lp>° <n>}}$

<lp> is the place for the reference tag; if omitted, the implied insert point is used
<n> is the current decimal label of the block to be referenced

This command puts a decimal label reference tag in the text at the specified point referring to the block whose decimal label is specified. As an example, the following display line:

```
     for more information, see .
```

and the command line:

```
     MDR/see /1.2.3
```

would result in:

```
     for more information, see %T '1.2.3' .
```

which, after Fullprinting, would appear as:

```
     for more information, see 1.2.3.
```

Note that if the block currently labelled 1.2.3 is moved such that its decimal label level changes, the tag will reflect this change.

• Make Decimal Reference Deferred

$\boxed{\text{\underline{MDRD}EF <lp1>° <lp2>}}$

<lp1> is the place for the reference tag; if omitted, the implied insert point is used
<lp2> is an <lp> of the block start of the block to be referenced

This command does exactly the same thing as MDREF, except that the second parameter is an <lp> of the desired block start, instead of its decimal label (see MDREF). This form of the command is typically used with the <lp> deferred to allow the user to travel through the file to find the <lp>. This should be used when the current decimal label of the desired block is not known, which is the usual case in dynamic, changing files.

• Make File

$\boxed{\text{MFILE <file> [<wind>°] <pass>°}}$

This command creates a FRESS file with the specified name and password. If <pass> is not specified, the password "DEFAULT" will automatically be used. The assignment of "DEFAULT" for <pass> will allow the file to be accessed in the future wihout password specification.

After issuing a MFILE, the user is placed in Input Mode [in the window specified; if none specified, current window is assumed] and can immediately start inserting text into the start of the file, which will be placed after the "*START OF TEXT AREA*" line.

• Move from Work Space

$\boxed{\text{MFWORK <lp>°}}$

<lp> is the point to move the work space text to; if omitted, the implied insert point is used

MFWORK allows the user to move ALL the text in his work space to a specified place in his file. The text from all areas in the work space is moved. However, the area lines themselves can not be moved, so a text-only edit (see Section 4.1.6) should be done. If the user desires to move only sections of the text in the work space, he should use the ordinary Move instruction with deferred <lp>'s.

• Make Jump

```
MJUMP <lp1> <lp2> <text1>° <text2>° <vs1>° <vs2>° <keys>°
```

<lp1> is the place for the jump to go
<lp2> is the place for the pmuj to go
<text1> is the explainer on the jump
<text2> is the explainer on the pmuj
<vs1> is the viewspec string on the jump
<vs2> is the viewspec string on the pmuj
<keys> is the keyword string for the jump and pmuj

The Make Jump command creates a jump in the user's file which the user may subsequently choose to take or not to take whenever he reaches the point in the file at which the jump was made. The explainers are literal, editable text strings which appear in-line within the file, to explain to the reader where he will go if he takes the jump.

Jump and pmuj viewspecs are optional. They are literal character strings specifying the viewspecs to be used if the jump or pmuj is followed. The viewspec string is the same as that used with the Set Viewspec command. It is limited to 255 characters in length. Jump viewspecs need not bear any relation to those on the pmuj.

Optional keywords may be used for temporarily turning Jumps into splices, for example, for linking files together for printout purposes; see SKJUMP and Section 2.5. The keyword string is limited to 255 characters and is described fully in Section 3.1.1. The jump and pmuj keyword strings are the same. Editing one string causes similar editing to be done on the other string.

• Make Label

```
MLABEL <lp>° <label>
```

<lp> is the place to make the label; if omitted, the implied insert point is used

Labels, as mentioned under GLABEL, provide a convenient means of identifying important points in a FRESS file which can later be accessed directly without having to pattern-scan for them. The label appears in-line in the text as "%L(LABEL)", and is entered into the Label Display Space, which is accessible through the DSPACE command. A label may be from 1 to 16 characters, may contain embedded blanks, and may be edited.

• Make Picture Reference

```
MPREFERENCE <lp>° <pict>
```

<lp> is the place for the picture reference tag; if omitted, the implied insert point is used

This command creates a tag which refers to the picture <pict>. The tag is displayed as

%T'name'

The picture itself can be displayed by jumping on the tag.

• Make Splice

```
MSPLICE <lp1> <lp2> <text1>° <text2>° <vs1>° <vs2>°<keys>°
```

<lp1> is the place for the splice
<lp2> is the place for the ecilps
<text1> is the explainer on the splice
<text2> is the explainer on the ecilps
<vs1> is the viewspec string on the splice
<vs2> is the viewspec string on the ecilps
<keys> is the keyword string for the splice and eclips

This command creates a splice in the file. As opposed to a jump, no option exists for the user when he arrives at a splice; the splice is automatically taken, i.e., it is unconditional, like a splice in a tape or film.

A typical use for a splice is to bypass some text that is no longer applicable but, for possible future reference, should not be deleted or placed in a separate file where it may be forgotten. A splice is made around the text being ostracized, and it is gone from view but still in the file. (The user might like to label it first so that he can access the portioned off area rather than deleting the splice to access it.) Another use of the splice is splicing files together for printout.

• Move to Label

$\boxed{\text{\underline{MT}LABEL <scope> <label>}}$

<scope> is the amount of text to be moved
<label> is the label after which to move the text

This command allows the user to move a piece of text to a position beginning immediately after the specified label. The command behaves precisely like MOVE in all respects and is useful for gathering diverse fragments at a common origin, similar to work space functions in the work space. MT does not work interfile.

• Move to Work Space

$\boxed{\text{\underline{MTW}ORK <scope>}}$

<scope> is the amount of text to be moved

A work space is available to the FRESS user (see Section 3.2.2). The MTWORK command allows the user to move a specified piece of text to the bottom of the last area in the work space. Within the work space, the user may perform all normal FRESS functions upon the text he has there.

When finished manipulating text in the work space, the user may return the entire collection to the main space of his file using the MFWORK command.

• New Area

$\boxed{\underline{\text{NA}}\text{REA <space>}^{o}}$

This command creates a new area (similar to a new piece of paper) in the space specified (either MAIN or WORK). If no <space> is specified, the area will be made at the bottom of the space currently being displayed.

After the command is executed, the display starts at the start area line of the new, empty area. The user may then input text or structure into this area.

Multiple areas are useful to conveniently separate unrelated pieces of text. Keeping material in separate areas rather than separate files has the advantage of allowing easier keyword and label retrieval.

O

• Offline Read (single window version only)

```
OREAD <options> <filename> <password>°
```

The OFFLINE READ command creates a FRESS file from one of two sources of input: 1) from cards in the virtual card reader, or 2) from some specified disk file. Input to the command is of two main types, card image input and line input. Card image input is input coming from cards in the virtual card reader or from card images in a fixed length disk file. Line input is input coming from a variable length disk file. Note that some of the options listed below apply only to input of one of the above types. The options may be specified in any order, separated by commas. Each option specified cancels any previously specified option with which it conflicts. Only the first character of each option is looked at. The following is a list of options:

Reader
: specifies that the input to the file is to come from the virtual card reader. No blank inserting is done after each card. That is, the text is considered to be one continuous stream, except that all trailing blanks except one on a card will be ignored. Blank cards will be ignored completely.

Disk[.<filename>[.<filetype>[.<filemode>]]]
: specifies that the input to the FRESS file is to come from the specified CMS file. All three parameters are optional and default to the FRESS filename, SCRIPT, and * respectively.

Lower
: specifies that the input to the file will be in both lower and upper case. This is the default input mode.

Upper
: specifies that the input is in upper case only, which means that the input will be translated to lower case and capitalized as indicated.

CapitalizeK
: specifies that the capitalize character will be K, which can be any character at all. Example: C* specifies that the capitalize character is to be a *.

In the input stream, a single capitalize character specifies that the next character is to be capitalized, which is useful when U is specified as an option, since everything is translated to lower case. A string of text enclosed by double capitalize characters is all capitalized. The default capitalize character is the %.

HyphenK

specifies that the logical hyphen character is to be the character K, which may be any character. The logical hyphen character pertains only to line input, and when placed at the end of an input line, causes no blank to be inserted after the line. This is useful when breaking a word across a line boundary. The default hyphen character is the %.

Startnn

specifies that input starts in column nn for a card image input item. The default starting column is 1. This option applies only to card image input.

Endnn

specifies that input ends in column nn for a card image input item. This option is useful for ignoring sequencing numbers at the end of a card. The default ending column is 80. This option applies only to card image input.

Bottom

specifies that the input is to go at the bottom of the current FRESS file.

The complete list of default options is: d,l,c%,h%,s1,e80 on an upper/lower case terminal, and d,u,c%,h%,s1,e80 on an upper case only terminal.

If the FRESS filename is not specified, it will be defaulted from the CMS filename if d is specified. If no filename at all is specified it will be defaulted to ((TEMP)). The file ((TEMP)) FRESS is erased before the new one is created. This feature is useful in terminal input mode, to use Offline Read to add a large block of text to an existing file. The text can be put in the temporary file with Offline Read and can then be moved to the real file.

If r is specified and the card reader is empty, the Offline Read command will wait for the card reader to fill before continuing. When this occurs, the message WAITING FOR CARD READER TO FILL will be typed.

The following is a list of error messages (with an explanation if necessary) issued by the Offline Read command:

STARTING COLUMN > ENDING COLUMN
STARTING COLUMN < 1
ENDING COLUMN > 80
FILE NOT FOUND - issued when CMS file specified does not exist.
FILE ALREADY EXISTS - issued when a FRESS file with the given filename already exists.

• Offline Type

$\boxed{\underline{\text{OT}}\text{YPE <space>}^{\circ}}$

This command will print on the offline printer the entire specified space exactly as it is displayed online (under the current viewspecs and line width). If no <space> is specified, the space currently being displayed is used. This command is useful for debugging one's format and structure codes. Note that the printed output will appear in upper and lower case even if an upper case terminal is being used.

<u>P</u>

<u>• Print (single window version only)</u>

```
PRINT <n>
```

This command is used to print online at most "one display buffer's worth" of text. TYPE or FULLPRINT must be used for longer online printouts. PRINT does not move the start of the display buffer. The maximum number of lines which may be printed is determined by dividing the buffer size (500 for a typewriter-like terminal, 700 for a display console) by the current line length (set by a SDISPLAY command). Thus with the default line length of 50 on a 2741, up to 10 lines may be printed.

<u>• Pack File</u>

```
PFILE <n>°
```

<n> is the percentage each page is to be filled

This function causes the internal structure of the current file to be altered such that each internal page or record is filled according to the percentage specified. The file must be open and is freed after packing. If no percentage is specified, 100% is assumed. No percentages below 50% are allowed. Internal FRESS pages are usually balanced at approximately 66% full.

A typical command would be:

```
PF 85
```

which would re-structure the current file so that each internal page would be as close as possible to 85% full. When the packing has been completed, the message 'PACK SUCCESSFUL' will be typed, followed by two numbers indicating the size of the file in pages before and after packing. Thus the response might look like:

```
PACK SUCCESSFUL 0034/0025
```

which would indicate the file had 34 pages before packing, 25 pages after packing.

PFILE is used primarily to make a file as small as possible when it is to be stored without editing for a long period of time. It is not advantageous to pack a file to 100% capacity if editing operations are still to be performed since any increase in amount of text at any point in the file will be likely to require the formation of new internal pages. However, one might want to pack a file to perhaps 85% capacity if only very minor typographical corrections were to be made subsequently.

• Print Picture

```
PPICT <pict>
```

This command adds the picture specified by <pict> to a file for Calcomp (offline plotter) output. The file is called PICTOUT and has filetype FRESSP. To plot the picture(s), the following commands should be issued from CMS:

```
PLOTPICT <acct number>
(wait for TAP1 ATTACHED msg)
hit BREAK
type B
```

After the plot is finished, the PICTOUT file is erased.

Q

• Query Current Files

```
QUERY <option>° [<wind>°]
```

The QUERY command is used to display the file name of and location in the user's current FRESS file (<option> = L, the default). The location is specified in terms of space name and area number. It can also be used to display the file names of all currently open files (<option> = F). This command is very useful when the user has forgotten the names of the files he currently has open, since the file name must be specified in order to free the file (see FREE FILE command).

[In the multiple window version, the Query is done for the specified window. If none is specified, the current window is assumed. However, a Q F will list all open files, not just those opened in that window.]

R

• Return

```
RETURN [<wind>°]
```

In the single window version, when a FRESS command causes the user's display to move nonlinearly (DSPACE, GLABEL, JUMP, BOTTOM, MOVE in Transcription Mode, GFILE, and MFILE commands) or if the user explicitly saves a display point (see SAVE), the system automatically "pushes" the current display point into a LIFO (last in - first out) stack called the Return Stack. Specifying RETURN causes the system to "pop" the stack and to relocate the display buffer at the popped point. Using the stack to mark his path through a text, the user can retrace this path.

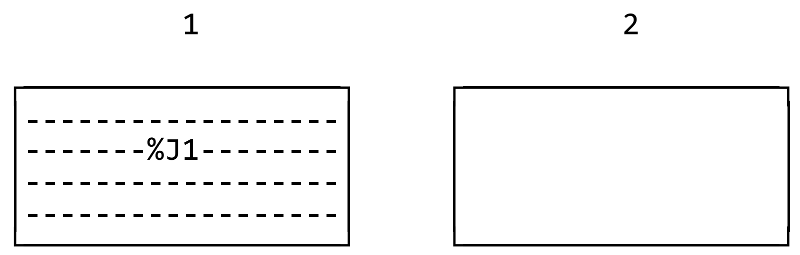
The operation of this stack is slightly different in the multiple window version. In this case, a display point is saved only if it is <u>replaced</u> by the results of a non-linear travel other than a RETURN. (This is always true in the single window version.) That is, if a GLABEL from window 1 causes the text which was in window 2 to be replaced by the specified label and following text, the display point from window 2 would be saved in the LIFO stack. It would make no sense to save the display point from window 1 in the stack, since it would still be visible on the screen.

If a window number is specified, the return point will be displayed in that window. If none is specified, the system will try to place it in the window in which it originally appeared. If this is not possible (i.e., the window no longer appears on the screen), another window will be used. In any case, the window in which the return point is displayed becomes the current window.

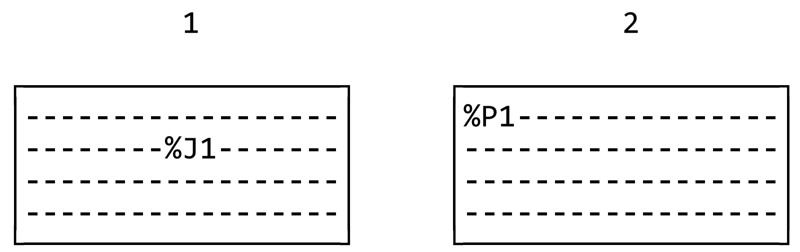
Two examples of the operation of the return stack in the multiple window version follow:

Example 1:

The current window is #1. Window 2 is blank.

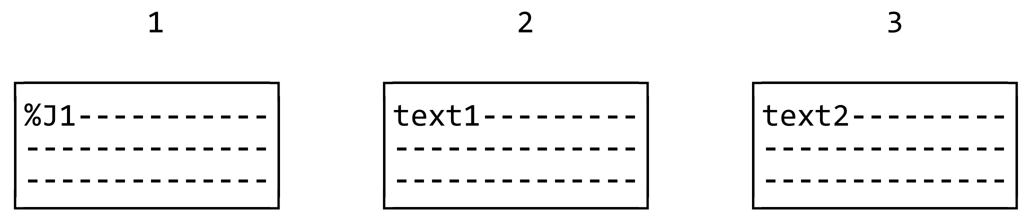


User types 'J/%'. Result is:

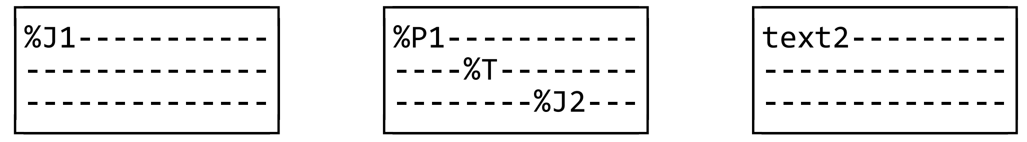


At this point, no locations have been saved in the return stack, as explained above. The command RET would produce the message 'NO MORE RETURN POINTS' and the display will remain unchanged.

Example 2:
The current window is 1.



User types 'J/%'. Result is:



User jumps on the tag in window 2 using lightpen (without changing current window). Result is:

```
%J1-----------    %P1-----------    %<|annotation-
--------------    ----%T--------    -------%>|----
--------------    --------%J2---    --------------
```

User jumps on jump in window 2 using lightpen. Result is:

```
%J1-----------    %P2-----------    %<|annotation-
--------------    --------------    -------%>|----
--------------    --------------    --------------
```

Notice that the new display replaced window 2, not the current window which is still 1. User now types 'ret'. Result is:

```
%J1-----------    %P1-----------    %<|annotation-
--------------    ----%T--------    -------%>|----
--------------    --------%J2---    --------------
```

The current window is now window 2. User types 'ret/1'. Result is:

```
text2---------    %P1-----------    %<|annotation-
--------------    ----%T--------    -------%>|----
--------------    --------%J2---    --------------
```

The current window is now window 1. In this case, the display point replaced in window 1 is not saved, since it was replaced on a Return. User now types 'ret/3'. Result is:

```
text2---------    %P1-----------    text1---------
--------------    ----%T--------    --------------
--------------    --------%J2---    --------------
```

Window 3 is the current window. There are no more return points now.

• Revert

```
REVERT
```

After performing a FRESS editing operation, the user is able to undo the edit by issuing REV. Only the last editing operation preceding the REV command may be undone, and ACCEPT neutralizes the ability to revert.

Traveling, both linear (scrolling, pattern scanning) and nonlinear (jumps, getlabels), may be done between an edit and a revert without negating the revert capabilities. Revert also reverts the starting point of the display buffer to the point in the file at which the now reverted editing operation was specified (or completed in the case of a deferred specification).

• Ring

```
RING <options>° [<wind>°]
```

The Memory Return Ring is designed to allow the user to travel easily and repeatedly among designated places he is interested in. This is different from the Return Stack which allows the user to backtrack over a single point only once.

There are five <options>:

ADD	Adds current location to ring.
DELETE	Deletes the ring entry currently in use, or the one most recently used, and starts the display at the next entry in the ring.
FORWARD	Starts display at next ring entry.
BACK	Starts display at previous ring entry.
CLEAR	Erases all ring entries.

If no <option> is specified, FORWARD is assumed.

[In the multiple window version, there is still only one Memory Return Ring. The window number, if specified, indicates either which display point to add to the ring or where to display the next entry. It has no meaning when DELETE is specified.]

• Refer To Annotation

```
RTANNOTATION <lp1>° <keys>° <lp2>
```

<lp1> is the place for the tag to go; if omitted, the implied insert point is used
<lp2> is the block to be referenced

A tag is made, in the indicated place, which references the indicated block in the annotation space. The block to reference may be indicated in one of two ways. If an <lp> is given, it is taken to be an <lp> of the desired block start. Alternatively, as in other commands, this <lp> may be deferred by using a question mark in place of the <lp>. A new block-start/block-end pair referenced by the new tag is placed around the indicated block. Any keywords specified will be placed on the tag and the block.

S

• Substitute Text

```
SUBSTITUTE <scope> <text>
```

<scope> is the text to be substituted for
<text> is the literal string to be substituted; if omitted, Input Mode is entered

Substitute allows the user to replace a string of text in a file by a new string. This command is useful for typographical corrections. For example, to change "useful" to "useless" in the previous sentence, the user could type

```
s useful useless
```

Substitute is also useful for editing longer strings of text with an ellipsis. Consider the following:

FRESS is an amazingly convenient text editor.

To substitute "a reliable" for "an amazingly convenient", the user need only specify:

```
s /a...nt/a reliable
```

See also USUBSTITUTE.

• Save Current Location

```
SAVE [<wind>°]
```

SAVE "pushes" the current display point [from the specified window; default is current window] into the Return Stack (see RETURN). It is useful if the user wants to look for some other point in his file (by scrolling or pattern scanning) and still be able to return easily to the original point. (Normally, only nonlinear traveling functions causes his original location to be remembered.) For example, specifying "SA>20>R" will bring the user back to his original viewing point after scrolling forward 20 lines.

• Scroll

```
<n>     or     SCROLL <n> [<wind>°]
```

<n> is the signed or unsigned number of lines to scroll

Scroll moves the display the specified number of lines of the current line length (see SDISPLAY). Scrolling may be done either forwards or backwards. To scroll forward, the user types just a number. Scrolling backwards is specified by preceding the number with a minus sign.

No command name need be specified to scroll. Examples:

```
  4 (travels forwards 4 lines)
-25 (travels backwards 25 lines)
```

NOTES: A scroll of zero lines just re-displays the current line. A more efficient way to re-display the current line is to type "P 1" to print one line (See Print). No spaces are allowed between sign and digits. A maximum of 127 lines may be scrolled at a time.

[In the multiple window version, the second format given above must be used if the user desires to scroll in other than the current window.]

• Set Display Window (single window version only)

```
SDISPLAY <n1>° <n2>
```

<n1> is the number of lines in the display window
<n> is the length of each line in the display window

Using this command, the user may specify the number of lines and the line length of the text which is displayed after each function is executed (assuming the DISPLAY mode is in effect; see section 4 and SMODE below). The default of the system for typewriter-like terminals is one line of 50 characters. The command

```
sd 2 120
```

would result in two lines of 120 characters being displayed every time a buffer was generated. The only restriction is that the line length times the number of lines must be less than the display buffer size (500 characters for typewriter terminals). See Appendix A on terminal characteristics for defaults and buffer sizes for other terminals.

• Scratch file

```
SFILE <file>
```

Scratch file erases the specified file. The file must be the current file in order to be scratched. The message "SCRATCH SUCCESSFUL" appears if the file existed and was properly scratched. Error messages are produced if the file was not current. A GFILE must be specified to access another of the open files. NOTE: This command can not be reversed using the Revert command.

• Set Keyword Annotation Request String

```
SKANNOTATION <bool>° [<wind>°]
```

This command establishes or deletes a keyword request string against which keywords on annotation tags are compared when the tag is encountered for display or printout. If the keywords on the tag satisfy the request string, the annotation block associated with the tag is displayed inline after the tag. When the end of the block is reached, the display returns to the text after the tag. In the case of printout, the text in the block will be printed as a footnote if the keywords match. Specifying SKA without a request string deletes the previous request so that no tags will be "followed".

[If a window number is specified in the multiple window version, the request string will be applied only to that window. If no window number is specified, it will be applied to all windows.]

• Sketch (Multiple Window Version)

```
SKETCH <pict>
```

This command loads and starts the IMLAC drawing package. The screen will display the word 'LOAD' while this is done. The parameter <pict> is the name of the picture; it must be 8 or fewer characters in length. If the picture is in the current FRESS file, it will be displayed and can be changed. If it does not exist, a new picture can be created and then added to the FRESS file. (To save time when working with several pictures, use window configuration 1B. This allows the Sketch program to remain in the IMLAC, so it will not be reloaded with each picture.)

• Set Keyword Jump Request String

```
SKJUMP <bool>° [<wind>°]
```

The SKJUMP command sets up a keyword string to be used in deciding whether or not to take keyworded jumps. When a jump is encountered while the file is being printed offline or displayed online, the keyword request string is compared to the keywords on the jump; if they match, the jump will be taken as if it were a splice.

The string may be replaced at any time with a new SKJUMP request. If the user wishes to eliminate his request string entirely, he may do so by specifying a null request string.

[In the multiple window version, the specified keyword string will be used in the specified window only. If no window number is specified, the string is used in all windows.]

• Set Mode of Display

$\boxed{\text{\underline{SM}ODE <option>}^{o}}$

SMODE allows the user to select which combinations of DISPLAY or BRIEF, and STATIC or TRANSCRIPTION modes he desires (see section 4 of the User's Guide).

<option> may be any string of the following symbols concatenated together (with no intervening blanks):

SYMBOL	MODE	MEANING
B	Brief	display off, nothing printed
D	Display	the amount of the display buffer specified by the last SD command (or system default) is printed after each command
S	Static	buffer does not move with editing operations (used primarily on display consoles)
T	Transcription	buffer moves to one word or code before the beginning of the editing operation performed (system default, used primarily on typewriter terminals)

"DT" is the default option in the single window version. [The default for the multiple window version on the IMLAC is DS.] The <option> string is interpreted letter by letter from left to right. Any invalid letters are ignored. In addition, conflicting modes (ST or BD) are resolved by using the right-most mode in the string. For example, if the user meant to set his viewspecs (SV) but typed "SM print" by mistake, the string "print" would set transcription mode because of the ending "t", while all other letters would be ignored, since they are invalid mode characters.

The mode set may be temporarily overridden by using the mode indicators at the end of a command line, after two blanks and a "." (see Section 4.3 of the User's Guide). To suppress display for one command, for example, just use this "blank blank period" convention.

• Scale Picture

$\boxed{\text{\underline{SP}ICT <pict> <option>}}$

This command sets the size at which a picture specified by <pict> will be displayed. The choices for <option> are N for normal scale, Q for quarter scale, H for half scale, or D for double scale. Normal size is the size of the picture when it was originally drawn. If a picture is changed by using the SKETCH command, the size will be reset to normal.

• Split area

$\boxed{\text{\underline{SPL}ITAREA <lp>}^{o}}$

<lp> is the split point; if omitted, the implied insert point is used

This command defines "areas" within the main text or work spaces of a file. An area in the main text space is bounded by *START OF TEXT AREA* and *END OF TEXT AREA* lines. When the SPLIT command is issued, the current area is "split" at the point of the <lp> and an *END OF TEXT AREA* line appears after the <lp> location. The user's display point is at the top of the second area (the one after the split point).

It is a good practice to have a label, splice or jump in or to the second area before issuing SPLIT. The only other way to access the area is through the structure space.

The difference between SPLITAREA and NAREA is that the former splits the current area in two, while the latter creates a brand new, empty area.

• Surround Text

```
SURROUND <scope> <lit1> <lit2>°
```

<scope> is the text to be surrounded

This command surrounds the specified text (regular text or explainers) with the specified literal text as follows. For the display line:

    text regular text or explainers

the command line:

    sur regular...ners ( )

would result in:

    text (regular text or explainers)

If <lit2> is omitted, <lit1> is used in both places.

• Set Viewspecs

```
SVIEW <vs> [<wind>°]
```

The VIEWSPECs determine how the text displayed online is presented and formatted. The system default (in the single window version) is for no online formatting, with all formatting codes beginning a new line and all structure codes (e.g., labels) shown in-line.

The viewspec string is in the format explained above in Section . The default for the single window version is EDIT viewspecs. └The default for the multiple window version is

    NORMAL-@-&-O+BL+CN

THIS produces a display which is pleasant to read but also allows editing operations. It also allows special character codes to be interpreted for the IMLACdisplay and does not show splices, ecilpses, or pmujs.]

[In the multiple window version, the viewspec string will take effect in the specified window. If none is specified, it will take effect in all windows.]

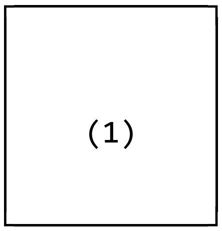
To obtain a short formatted printout of a section of a file, the user should locate the section and then do a "sv print" followed by a "type 100" to see 100 lines.

• Set Window Configuration (multiple window version only)

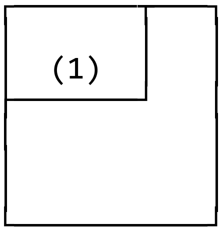
```
SWINDOW <option>
```

This command sets up one of 7 default window configurations. The following drawings illustrate the window settings, which are named 1A, 1B, 2A, 2B, 3A, 3B, 4A.

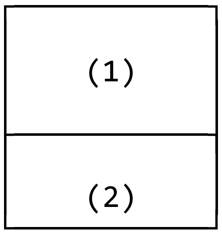
1A



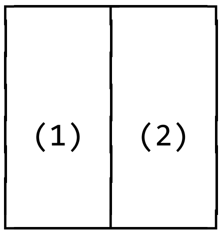
1B



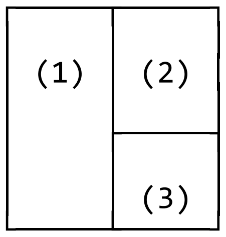
2A



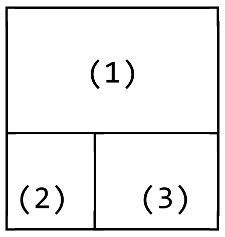
2B



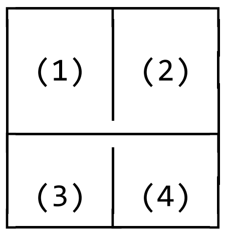
3A



3B



4A



<u>T</u>

<u>• Type (single window version only)</u>

$\boxed{\underline{\text{T}}\text{YPE <n>}}$

<n> is the number of lines to be typed

The TYPE command allows the user to have any number (less than 1000) of lines typed at his console. Unlike PRINT, the <u>last</u> printed line is the start of the display after the completion of the command.

<u>• Top Input</u>

$\boxed{\underline{\text{TI}}\text{NPUT <space>}^{\text{o}}}$

This command causes input mode to be entered at the top of the specified space. Valid spaces are <u>t</u>ext and <u>w</u>ork. If no space is specified, the current space is assumed.

• Trail

```
TRAIL <option>° [<wind>°]
```

TRAIL is used for going from one block to another in a block trail (see BT and BTD). It must be used for stepping through a discrete trail. If the trail is left in the middle, issuing the TRAIL command will rejoin the trail where it was left.

<option> is the direction, is either "f"orward or "b"ackwards. If <option> is not specified, "f" is assumed.

[If a window number is specified in the multiple window version, the next block will be displayed in that window. If none is specified, a window other than the current window is chosen.]

U

• Underscore

```
UNDERSCORE <scope>
```

<scope> is the amount of text to be underscored

The UNDERSCORE function inserts underscore format codes (i.e., !(0...!)) around the specified text.

• Uncapitalize

```
UNCAPITALIZE <scope>
```

<scope> is the character string to be put in lower case

The specified string will be put in lower case. However, the first non-blank character after any "sentence-ending" punctuation (period, question mark, literal exclamation point) or after any format (macro, alter, or edit) code will be capitalized so the <scope> may include multiple sentences where the first character of the sentence is to be capitalized. If the <scope> does not include the format code or the sentence-ending punctuation, the following letter will not be capitalized. Thus given the string

```
!-p-THE QUICK BROWN FOX
```

the command

```
UNC/!...X
```

would result in

```
!-p-The quick brown fox
```

but the command

```
UNC/TH...X
```

would result in

```
     !-p-the quick brown fox
```

Note that this command ignores the original case of the string, so it may find a string which is already in lower case.

• Uniform Substitute

```
USUBSTITUTE <scope> <text> <options>°
```

<scope> is the amount of text to be substituted for
<text> is the character string to replace it

UNIFORM SUBSTITUTE allows the user to repeat a normal text substitute an arbitrary number of times. The user can specify that the substitute should be made a specified number of times or uniformly throughout the current file, he can optionally accept or reject each substitute and escape from the command at any time.

The first two parameters are the same as for SUBSTITUTE. The <options> parameter determines how the uniform substitute is to be executed and displayed. The allowable options are:

I	Inquire after each substitution
A	perform each substitution Automatically
D	use Display mode to show the effect of each substitution
B	use Brief mode to suppress display of each substitution
L	do a Long scan for the <scope> pattern
n	perform the substitution "n" times only

The inquire option means that the user must explicitly accept or reject each substitution in order of occurrence, and may exit from the command at any time. The automatic option means that the substitution proceeds without user intervention, and ends only when no further occurrence of <scope> remains. Option "n", if used, must be the last option in <options> and should be less than 4095. An important note, regardless of which options are in effect, is that the REVERT command can not be used to undo a UNIFORM SUBSTITUTE.

As an example, if the user is in display and transcription modes, specifying:

```
us/FRESS/the editor/a5
```

will automatically substitute "the editor" for the first five occurrences of "FRESS" starting at the current buffer location, and displaying each line where the substitution occurs.

If no <options> are specified in the command, Uniform Substitute will use defaults of Inquire, Display, and a count of 4095. If <options> are specified, these options will be used in addition to the static/transcription mode currently being used (see SET MODE command). The <options> string is scanned from left to right. If conflicting options (e.g., AI) appear in the string, the rightmost option is used.

In Inquire mode, after each replacement, "OPTIONS=" will be displayed at the terminal. The user then responds with either A (accepting the substitute) or R (rejecting it). Immediately following A or R, the user may type E to exit from the UNIFORM SUBSTITUTE command and return to normal Command Mode. If E is not specified, the user can optionally follow the A or R with new <options> characters. If the L option is not specified, only 2100 characters will be scanned for <scope>. If the pattern is not found, the message

```
PATTERN NOT FOUND, OPTIONS=
```

will be typed. The user may reject the command (E or R) or continue with a long scan to the end of the file (A).

5.6 FRESS HOUSE FUNCTIONS

A certain group of commands are called "house" functions (because they perform "housekeeping" functions for the command language interpreter). These commands are specified by using an ampersand (&) followed by the name of the command. As with other FRESS commands, only the first few letters of the command name (as shown by underscores below) are needed. If an operand is necessary, the command name is followed by one blank and then the operand. House functions may not be stacked on command lines using the FRESS command seperator (>).

• Enter Asis Mode

```
&ASIS
```

Issuing this command causes the following to happen to input values (lines in Input Mode, and all <text> parameters): each is preceded by !-s0- and each occurrence of the tab key character (from the physical tab key) is replaced by !-T-. See Section 5.11 of the User's Guide for the use of &A with table formatting.

• Clear Function Area

```
&CLEAR
```

Issuing this command clears all pending functions and deferred <lp>s and <scope>s. It should be used if the user has a function with a deferred <lp> or <scope> and decides that he does not want to complete the function.

• Execute

```
&EXEC <file>
```

The &EXEC command gives the user the ability to execute a sequence of FRESS commands by typing a single command at the terminal. The sequence of commands is taken from the CMS file <file> of filetype MEMO created with the CMS editor. For an explanation of this editor, see the CMS User's Guide and the Interactive User's Guide.

The format of the lines in the MEMO file is similar to the format of command lines typed individually into FRESS. Only one command should appear on each input line, unless the FRESS logical command separator (>) is placed between commands on the same line. Any FRESS command, including other house functions, may be included in a MEMO file processed by &EXEC. A blank line is used to pass from INPUT mode to Command Mode while using &EXEC. Errors will be handled as usual, but no user intervention is allowed (to correct errors) until all commands in the file have been executed. Therefore, the user should exercise great caution when he creates the MEMO file. Reading from the file continues until all commands have been processed or an error in reading from the MEMO file occurs. The user is informed of either of these events by a message typed at the terminal. Normal FRESS processing (i.e., normal terminal input/output) then resumes.

As an example, if a user frequently wants to insert a decimal block at the bottom of his file, he might set up a MEMO file containing the following three lines:

```
BI

IDB/
```

The first line would put the user into Input Mode at the bottom of the file, thus setting up an implied insert point there. The second line, a null ine, would put the user back into command mode; the third line would insert a decimal block at the implied insert point, which is the bottom of the file, and put him in Input Mode inside it. Thus the user can, by suing the &EXEC feature, execute all three lines by merely typing one.

• Repeat Last Command Line

$\boxed{\text{\underline{\&G}IN <n>}^{o}}$

<n> is the number of times to repeat

This command repeats the entire last command line (which was not an '&g') for the specified number of times. If no operand is specified, '1' is assumed. This function is useful for iterating a pattern scan, a substitute, or a delete. If an error is encountered during the iteration, execution of the command halts immediately.

An example of this command follows. If the user is reading a small section of text, he might print 10 lines, read them, then desire to scroll forward and print the succeeding 10 lines. The sequence of commands would be

```
p␢10
10>p␢10␢␢.*
&g
```

The first command prints 10 lines. The second command line scrolls 10 and then prints 10 lines with only the last command on the line causing a display (see Section 4.2 in FRESS User's Guide). After reading those ten lines, the user would then type the third command line above, which would repeat the scroll and the print.

Note that the CMS linend character (defaulted to #) marks the end of a logical command line although multiple commands may physically be on the same line. Thus if the user typed the sequence

```
10#p 10
&g
```

only the Print command would be repeated by the &G.

• Leave Asis Mode

```
&NORMAL
```

This command causes input values to be treated normally.

• POP Implied Insert Pointer

```
&POP <n>°
```

<n> is the number of pointers to pop from stack

The specified number of implied insert pointers at the top of the stack of pointers (see Section 2.6) are "popped" (removed) and the pointer below them becomes the current pointer. If there is only one pointer in the stack, as with ordinary non-blocked text editing, an error message is returned and the pointer stays in effect (see Section 2.6.2.5). If no number is specified, 1 is assumed.

• Route

```
&ROUTE <filename>° <filetype>° <"c">°
                    or
&ROUTE OFF
```

&ROUTE allows the user to save the output of his terminal session, or any part of his session, in a CMS file. The filename and filetype are optional and default to FILE FRESSOUT if not specified. If "c" is specified, only commands typed by the user are routed to the file. If "c" is not specified, FRESS responses will be routed as well. If the user wishes to specify any of the optional parameters, he must also specify any preceding optional parameters. For example, if he wishes to specify the "c" option, he must also specify a filename and filetype. The file may be printed using the CMS command Offline Print CC (o print cc <filename> <filetype>).

To halt routing, the user should type "&R OFF".

• Set Special Character

$\boxed{\text{\underline{\&S}ET <option> <character>}}$

FRESS has certain special characters which have special meanings at certain times to the command language interpreter. The user may redefine any of these by means of the &SET command if, for example, his text requires an excessive usage of that character.

The <option>s and their default characters are:

OPTION	CHAR	FUNCTION
POINT	|	location pointer delimiter
BREAK	>	command separator
KILL	<	no display for this command only
ATTN	&	"house" function delimiter
XSPOT	&	pattern locator
ELIP	...	pattern ellipsis
QUAL	-	command qualifier
BLANK	_	blank indicator

• Type on,off

$\boxed{\text{\&TYPE <option>}}$

<option> = ON or OFF

&TYPE is particularly useful with &EXEC. By specifying &TYPE OFF, all typing at the terminal is suppressed, but normal processing continues. Thus, &TYPE OFF entered before &EXEC <filename> will process the commands in the MEMO file but will not type the effects of the commands (including errors) at the terminal. At the termination of the &EXEC function, the type is automatically turned back on (if it was off) in order to let the user know that he must now reassume control. The type may also be turned on manually by entering &TYPE ON.

APPENDIX A: TERMINAL CHARACTERISTICS

There are two general types of terminals used for the single window version of FRESS - those with upper and lower case characters and those with only upper case characters. Terminals with both upper and lower case are used exactly as described in this manual. Their default display window is a single 50 character line and the buffer size is 500 characters.

Upper case terminals are used slightly differently. The correct version of FRESS to be used on these terminals is invoked by typing

```
FRESS T
```

The default display window in this version is a single 70 character line and the buffer size is 700 characters. When displaying a file or inputting text, those letters which are meant as capital letters should be preceded by a percent sign. Multiple upper case letters should be surrounded by double percent signs. All other letters are considered to be lower case. Thus, the sentence which would appear on an upper/lower case terminal as:

```
The quick brown fox
```

would appear on an upper case terminal as:

```
%THE QUICK BROWN FOX
```

Similarly

```
This text editor is FRESS.
```

would appear on an upper case terminal as:

```
%THIS TEXT EDITOR IS %%FRESS.%%
```

Numbers and punctuation (except exclamation points) may be considered upper case characters when preceded by 2 or more upper case letters. Thus the period above is part of the "multi-cap" string, but would not be considered an upper case character in the string

```
%P.%T. %BARNUM
```

Care must be taken when pattern scanning on an upper case terminal using a Locate or when specifying any context string. The pattern must exactly match the way the text appears when displayed, including the percent signs indicating upper case. Thus the string

```
%SENATOR %MC%GOVERN
```

would not be found if the user specified

```
L/MCGOVERN
```

It would be found by

```
L/MC%G
```

There is another difference in the operation of the Locate command between the two types of terminals. On an upper/lower case terminal, 5 consecutive Locates of the same pattern will always find 5 different instances of the pattern. On an upper case terminal, 5 consecutive Locates of a pattern beginning with a capital letter will find the same instance 5 times.

On upper/lower case terminals, labels, macros, and format codes are displayed in the case in which they are entered. On upper case terminals, however, there is no distinction made when these three kinds of text are displayed. That is, no percent or double percent signs appear to indicate upper case characters. Since it is impossible to differentiate between upper and lower case, it is generally wise, where the case is important, to input labels and macros in only lower case when an upper case terminal is likely to be used.

It often happens that an error is made and a section of text is entered in the wrong case, either by neglecting to use the proper version of FRESS or by forgetting the percent signs to indicate upper case. In these instances, the Capitalize and Uncapitalize commands may be used to correct the error. If, for example, an entire file is inputted in upper case accidentally, the Uncapitalize command, specifying the whole file as its <scope>, will flip all characters to lower case except those appearing after "sentence-ending" punctuation - periods and (literal) exclamation points.

Since the percent signs indicating upper case are not actually part of the file, it is not possible to capitalize characters by inserting percent or double percent signs, or by surrounding a text string with double percent signs using the Surround command.

Certain "special characters" do not appear on all types of terminals. Thus a not sign (¬), which does not appear on an ASCISCOPE, can be inputted in the 'T' version of FRESS as a backslash which is a "capital L" on the keyboard. Similarly an or bar (|) can be inputted as a circumflex, which is a "capital N". Any character which does not appear on the keyboard may be inputted as an exclamation point followed by the decimal number representing that character code. A list of some of these codes appears in Section 5.8 of the User's Guide.

APPENDIX B: LISTS OF COMMANDS ACCORDING TO TYPE

The following lists group FRESS commands according to the type and order of their parameters as explained in Section 5.4.2.

All parameters required

Add Password	AP
Bars	BA
Capitalize	CA
Copy File	CF
Copy	CO
Copy Picture	COP
Change Password	CP
Change Picture Name	CPI
Copy to Label	CT
Copy to Work	CTW
Change Current Window	CW
Delete	D
Delete Password	DP
Delete Picture	DPI
Free File	F
Insert Before	IB
Move	M
Move To Label	MTL
Move To Work	MTW
Print	P
Print Picture	PP
Substitute	S
Scratch File	SF
Sketch	SKE
Scale Picture	SP
Set Window Configuration	SW
Type	T
Underscore	U
Uncapitalize	UN
Execute	&E
Monitor	&M
Set Special Character	&S
Type On,Off	&T

All parameters optional

Bottom	B
Blank Window	BW

Bottom Input	BI
Block Trail Continuous	BT
Block Trail Discrete	BTD
Copy From Work	CFW
Display Space	DS
Fullprint	FU
Locate	L
Move From Work	MFW
New Area	NA
Offline Type	OT
Pack File	PF
Query	Q
Return	R
Ring	RI
Save	SA
Set Keyword Annotation Request String	SK
Set Keyword Jump Request String	SKJ
Set Mode	SM
Splitarea	SPL
Top Input	TI
Trail	TR
Repeat Last Command Line	&G
Pop Implied Insert Point	&P
Route	&R

Leading Required parameters

Footnote	FO
Get File	GF
Get Decimal Label	GDL
Get Label	GL
Jump	J
Make Annotation	MA
Make Block	MB
Make Decimal Block	MDB
Make File	MF
Make Jump	MJ
Make Splice	MS
Offline Read	O
Scroll	SC
Surround	SUR
Set Viewspecs	SV
Uniform Substitute	US

Trailing required parameter

Insert	I
Insert Annotation	IA
Insert Block	IBL

Insert Decimal Block	IDB
Make Decimal Reference	MDR
Make Decimal Reference Deferred	MDRD
Make Label	ML
Make Picture Reference	MPR
Refer To Annotation	RTA
Set Display	SD

<u>No</u> <u>parameters</u>

Accept	A
Display Viewspecs	DV
End	E
List Pictures	LP
Revert	REV
Enter Asis Mode Is	&A
Clear Function Area	&C
Leave Asis Mode	&N

The following FRESS commands use the implied insert point:

Copy From Work	CFW
Insert	I
Insert Annotation	IA
Insert Block	IBL
Insert Decimal Block	IDB
Make Decimal Reference	MDR
Make Decimal Reference Deferred	MDRD
Move From Work	MFW
Make Label	ML
Make Picture Reference	MPR
Refer To Annotation	RTA
Splitarea	SPL

The following FRESS commands are Editing commands:

Accept	A
Add Password	AP
Bars	BA
Bottom Input	BI
Capitalize	CA
Copy File	CF
Copy From Work	CFW
Copy	CO
Copy Picture	COP
Change Picture Name	CPI
Copy To Label	CT
Copy To Work	CTW
Delete	D
Delete Password	DP
Delete Picture Name	DPI
End	E
Free File	F
Footnote	FO
Insert	I
Insert Annotation	IA
Insert Before	IB
Insert Block	IBL
Insert Decimal Block	IDB
Move	M
Make Annotation	MA
Make Block	MB
Make Decimal Block	MDB
Make Decimal Reference	MDR
Make Decimal Reference Deferred	MDRD
Make File	MF
Move From Work	MFW
Make Jump	MJ
Make Label	ML
Make Picture Reference	MPR
Make Splice	MS
Move To Label	MTL
Move To Work	MTW
New Area	NA
Offline Read	O
Pack File	PF
Revert	REV
Refer To Annotation	RTA
Substitute	S
Scratch File	SF
Sketch	SKE
Scale Picture	SPI
Splitarea	SPL
Surround	SUR
Top Input	TI

Underscore	U
Uncapitalize	UN
Uniform Substitute	US

The following FRESS commands are Travel or Display commands:

Bottom	B
Blank Window	BW
Block Trail Continuous	BT
Block Trail Discrete	BTD
Change Password	CP
Change Current Window	CW
Display Space	DS
Display Viewspecs	DV
Fullprint	FU
Get File	GF
Get Decimal Label	GDL
Get Label	GL
Jump	J
Locate	L
List Pictures	LP
Offline Type	OT
Print	P
Query	Q
Return	R
Ring	RI
Save	SA
Scroll	SC
Set Display	SD
Set Keyword Annotation Request String	SK
Set Keyword Jump Request String	SKJ
Set Mode	SM
Set Viewspecs	SV
Set Window Configuration	SW
Type	T
Trail	TR

TABLE OF CONTENTS