FILE: FRESSPLM

Compliments of FRESS

A File Retrieval and Editing SyStem

Release 9.1   2 MAY 79

0. PREFACE

This manual is meant as a reference guide to the internals of FRESS. The programmer should be familiar with the externals of FRESS (e.g., by reading the Users' Guide and Guide B) before attempting to read this. It is hoped that the material is presented in a readable manner, but this is not a textbook which can be read sequentially. In other words, it's like any other manual - you have to read all of it to understand any of it.

1 OVERVIEW OF THE SYSTEM

2 FILE HANDLING

2.1 GENERAL PAGING DATA STRUCTURE

The data in FRESS files is stored on 1600 byte pages numbered consecutively from zero. The CMS block size is 800 bytes, so there are 2 CMS records per FRESS page. The pages are logically connected according to spaces: that is, the main text space pages, annotation space pages, etc. are each connected in a doubly linked list. The first page in the space will have a backward pointer (PREVPAGE) of zero; the last page will have a forward pointer (NEXTPAGE) of zero. The first and last page numbers (or page names) of each space are listed in a dictionary which appears on page 0, in the field PCBDICTS in the FCB. This is a 48-byte field, 2 halfword page names for each of 12 spaces. (Spaces 4, 5, and 6 are not defined, but there is room for them in the dictionary.) Each space has two dictionary numbers which indicate the position in the dictionary of the entries for the first and last pages in the space, respectively. The beginning dictionary number is

(space number * 2) - 1

and the ending dictionary number is

space number * 2

Thus the main text space, space 1, has dictionary numbers 1 and 2; the annotation space, space 2, has dictionary numbers 3 and 4, etc.

If a page in a FRESS file is deleted, it is removed from the logical page chain in its space and added to the list of free pages. This is a forward linked list originating in the PCB; the field PCBFLIST contains the page name of the top of this list. Each page in the list contains the page name of the next item in the list in the position normally occupied by the forward page chain pointer (NEXTPAGE). If a new page is requested, pages in this list are used before new blocks are created in the file.

In addition to these logical page connections, there are run-time links between pages that have been read from disk into core. Unlike the logical page connections, which are in terms of page names, these connections are absolute pointers to core locations. The pages from each open file are linked in a doubly linked ring which is "anchored" in the FCB for that file. (It is anchored in the sense that one finds the ring by using the appropriate fields in the FCB. A ring has,

of course, no real end.) These FCB's are also linked in a doubly linked linear list which is anchored at FCBLIST in the UCB. In both cases, the lists are in the order of most recently used item first. In the case of the in-core page chain, the least recently used item can be retrieved by going backwards in the ring from the PCB. The forward and back pointers in the in-core page chain are the 8 bytes at the top of each page (except in the FCB) and are not written out to disk. Thus a page on disk is 1600 bytes long, but a page slot in core is 1608 bytes. Displacements into a page are measured from before these pointers. (As it turns out, everyone considers a page to be 1600 bytes long, even in core, with the result that the last 8 bytes of a page are never used.) Free page slots are in a forward linked list starting at PAGEPOOL in the UCB. The first byte of this pointer, PAGECNT, is the number of free page slots in the list.

Whenever a routine wishes to ensure that a particular page remains in core, it turns on HOLDFLAG in HEADFLAG in the page header. HOLDFLAGmust be turned off when the routine no longer needs the page. If a routine changes something on any page, including the PCB, it must turn on MODIFY in HEADFLAG. Routines must be careful not to change the HEADFLAG set by someone else, so any routine which might do this (e.g. UPDTDSAR) must save the HEADFLAG before changing it and restore it before returning to the calling routine.

In addition to in-core page slots, pages are also sometimes temporarily stored in a spill file on disk with the file name SYSUT6. This is necessary because of the "revert" facility in FRESS. Each editing change must be temporary until the next edit is made or it is explicitly accepted by the user. Thus no altered pages can be written back into the original file until the editing change is made permanent; if there are no more page slots into which a page can be read, an altered page must be dumped to this spill file to free up its page slot.

Each FCB has a pointer to its own spill table, if one exists; this is in the SPILLPTR field contained in FCBFDCB in the FCB. The spill table consists of a halfword containing the number of entries in the table, followed by a fullword entry for each page of that file dumped to the spill file. These entries are a halfword pagename of the page in this file, and a halfword block (record) number of its position in the spill file.

There is an FCB for the spill file if it exists; the pointer to it is SWPDCBAD in the UCB. This FCB is not equivalent to other FCB's. It contains only the section of the normal FCB called FCBFDCB. All files share the same spill

file. The FCB of the spill file keeps track of how many blocks have been allocated and how many are currently being used. If this latter field is decremented to zero (which happens when a revert is done or an edit is accepted), the spill file FCB is freed. If the file is reused later, it will again be used starting at record 1.

Following is a list of the fields in the FCB control block with an explanation of each.

```
FCBFWDPT  DS      A         -> next FCB in linked list
FCBBKWPT  DS      A         -> previous FCB in linked list
FCBFILEN  DS      H         file number
FCBPCBIL  DS      H         disp from top of FCB to internal
                               label space dictionary
FCBPCBKY  DS      H         disp from  top of FCB to keyword
                               dictionary
FCBPGSZE  DS      H         page   size  (currently  set  at
                               1600)
FPROTECT  DS      CL40      protection  field  currently  in
                               effect
          ORG     FPROTECT
FPASSWRD  DS      CL7       password text string
FPROTKEY  DS      C         function byte
FPROTFUN  DS      CL32      function mask
          DS      0F
FCBFDCB   DS      CL88      internal   to  paging;  includes
                               read/write      and     state
                               parameter  lists;  read/write
                               flag; spill file info and ptr
                               to spill table
FCBFNAME  EQU     FCBFDCB+8   file name
FCBBLOCK  DS      F         unused
FCBPCB    DS      0F        start  of PCB  entries from page
                               zero
PCBFWDPT  DS      F         -> next page in page chain
PCBBKWPT  DS      F         -> prev page in page chain
PCBFLAGS  DS      C         header   flags   -   see  HEADER
                               control  block.  In  addition
                               has  flag  defined  in paging
                               meaning  this is  the head of
                               the page list.
PCBUSECT  DS      C         unused
PCBFREEB  DS      H         unused
PCBMAXBL  DS      H         #  of  blocks actually allocated
                               on disk
PCBFLIST  DS      H         page name  of start of free page
                               list
PCBNPAGE  DS      H         page  # of  last block allocated
                               as FRESS page
PCBGUYID  DS      CL8       was unused; now first 2 bytes is
                               next available internal label
PCBDICTS  DS      CL48      space dictionaries
PCBINLAB  DS      CL60      internal label space dictionary
PCBKYLAB  DS      CL362     keyword space dictionary
FCBFORMT  DS      2AL2      length,  disp  of format control
                               block (see ONLINE)
```

Following is a list of the fields in the HEADER control block with an explanation of each. Note that the same general format is followed on all pages, but there are some differences between spaces.

```
FOREPAGE  DS       F        -> next in-core page in page
                               chain
BACKPAGE  DS       F        -> previous in-core page in page
                               chain
HEADFLAG  DS       H        flags about this page:
MODIFY    EQU      x'80'    1 => this page was modified
HOLDFLAG  EQU      x'40'    1 => don't bump this page
ONLINFLG  EQU      x'01'    used to tell KEYWORDS not to
                               release the page since
                               KEYEDIT wishes it to be kept.
                               Otherwise KEYWORDS ignores
                               HOLDFLAG if set.
PAGENAME  DS       H        name (number) of this page
PREVPAGE  DS       H        previous logical page in space
NEXTPAGE  DS       H        next logical page in space
SPACENO   DS       X        space number of this page
          DS       X        free
FREEAREA  DS       H        disp from page top to free area
                               on page
DATAAREA  DS       H        disp from page top to start of
                               data area

* used in the main, annotation, and work spaces
AREANUMP  DS       H        number of the current area at
                               top of page
LASTINTP  DS       H        internal label of last structure
                               on previous logical pages
TEXTSCOP  DS       X        unused

* used in internal label space
LASTAREA  EQU      LASTINTP intlab of last area order of
                               file

* used in the keyword space
REFFLAG   EQU      AREANUMP byte of flags
OVERFLOW  EQU      x'01'    references split over page
                               boundary onto next page

* used in dslink space
CODEAREA  EQU      FREEAREA disp to dslink code area
NAMEAREA  EQU      DATAAREA disp to dslink name area
CODESTRT  EQU      NAMEAREA+2-HEADER  start of data area on
page

* used in the picture space
```

```
PICTLEN   EQU      AREANUMP  total length of picture which
                              starts on this page
```

2.2 PAGING ROUTINES

Most of the file handling routines exist as entry points in the csect PAGING. The exceptions to this are GETFCB, FREEFCB, PROTECT, and RESTORE. Except for some shared code for returning to the calling routine, the functions contained in PAGING are independent from one another and so will be treated as separate programs in this discussion.

The next page contains a macro level flowchart of the paging system. The routines which are underlined appear in the PAGEROUT table. In addition to the pictured routines, GETUCB, FREEUCB, and IEAFARTS (WIPEFSCB) also appear in PAGEROUT. Since they are discussed elsewhere (see section on storage management) they are ignored here.

The next sections are meant as general flow of control analyses of paging functions, not as exhaustive discussion of each routine. For more detailed analysis, see the individual module sheets in appendices.

NOTE: All paging routines assume that their function is to be performed on the current file as pointed to by FCBPTR in the UCB which is set by MAINLINE or GETFCB.

2.3 FINDING A FREE PAGE SLOT

Whenever a paging routine needs a free page slot, it calls FINDPAGE. FINDPAGE is internal to PAGING and thus is only called by routines in that csect. There are many ways to get a free page slot, but they are searched in a particular order:

1) a free page slot from PAGEPOOL in the UCB;
2) an unheld, unaltered page in the current file, which can be overlayed;
3) an unheld, unaltered page in another file, which can be overlayed;
4) an unheld, altered page in another file (call to SPILLOUT to find such a page and dump it to the spill file)

j8-5) ABEND if none of these schemes works

All searching in page chains is done backwards through the lists to get the least recently used page first. If a slot is found, it is linked in the front of the page chain from the current FCB.

2.4 CREATING A FILE

A file is created by calling GETFCB and MAKEFILE. For the user 'Make File' (MF) command, the calls are done by MAINLINE. They are also done from routines, such as PFILE and COPYMAIN, which create new files internally.

GETFCB creates an FCB for the file and initializes the file independent fields. It makes this FCB the current FCB by setting FCBPTR in the UCB to point to it. MAKEFILE sets up the rest of the FCB, including the PCB. It sets up page 1 as a protect page, page 2 as the first internal label page, page 3 as the first structure space page, and page 4 as the first main text space page, and writes each of them, including page zero (the PCB) out to disk.

2.5 GETTING A FILE

"Getting" a file means making that file the current file. This can be done either when the file is already open, that is, there is already an FCB for it, or when it is not open. In either case, the first thing done is a call to GETFCB. If the file is already open, GETFCB sets FCBPTR in the UCB to point to its FCB. In many cases this is sufficient. However, if the file is being opened, some additional initialization is necessary. In this case, GETFCB will create a new FCB and put it at the top of the list of FCB's. FCBPTR will be set to point to it. PAGEINIT will then be called to initialize the rest of the run-time paging environment. It sets up the FCBFDCB section of the FCB, moves the PCB in from page zero, and calls PROTECT to set up the protection fields.

2.6 CREATING AND DELETING PAGES

If a routine needs a new page in a file it calls CREATE. CREATE first checks the freed page chain beginning at PCBFLIST in the FCB to see if a page already allocated on disk can be

used. If not, it checks to see if a page has been allocated on disk but never used in FRESS. This is the case if PCBNPAGE is less than PCBMAXBL (which is EQU'd in paging to MAXBL). In the CMS version, this happens only if an editing change which increased the size of the file is reverted. Then PCBMAXBL has been updated, and the file can not be made smaller again, but the newly created page or pages have not been used in the reverted file. (In the MVT version, a size is specified when the file is created, and the number of blocks equal to that size is allocated immediately. When the number of pages actually used by FRESS equals the number of blocks originally allocated, a new and larger file must be created and the old file copied into it.)

If no previously allocated pages can be used, a new block is allocated on disk and PCBMAXBL is updated.

To delete a page, a routine calls DELETE (in PAGING, to be differentiated from the editing routine DELET), which adds the page at the top of the freed page list beginning at PCBLIST in the FCB. The removal of the freed page from the logical page chain of used pages in the file is done in the calling routine.

2.7 RETRIEVING A PAGE

GETPAGE is called to retrieve a page once the page number is known. It will first check to see if the page is already in core. If so, it will just rearrange the page chain to put the requested page first. If not, a free slot will be found, first by searching for a free page slot from the list originating in the UCB, then for an unheld, unaltered page from the current file, finally by calling FINDPAGE. The desired page will then be read into the page slot, from the spill file if the page is there, or from the original file. Note that when the RDBUF is done, the record number in the parameter list is 1 greater than the FRESS page number. This is because FRESS pages are numbered from zero, records are numbered from 1.

After the page is read into core, the page chain for the FCB of the file is arranged so that the requested page is first.

If a page must be held in core when other pages are to be read in, the HOLD flag must be set in the page header. This insures that GETPAGE will not bump the page when looking for a free page slot, thus invalidating pointers into that page.

Thus any routine which calls other routines should turn this flag on in all pages which it wants to keep. The flag must be turned off when the page is no longer needed. In addition, some routines, particularly in editing, must be careful not to destroy the flags set up by a routine which called it. For example, if routine A holds a page, then calls routine B which sets and then clears the hold flag, A still thinks the page is held, but it is not.

2.8 DICTIONARY HANDLING

To retrieve the first or last page of a space, a call is made to GETDICT giving the dictionary number desired (see section on space dictionaries - paging data structure). GETDICT returns the corresponding page.

To change an entry in the space dictionaries, a call is made to DEFDICT giving the dictionary number and the page number which should be placed in that entry.

NOTE: GETDICT and DEFDICT are multiple entry points to the same section of code in PAGING.

2.9 MAKING CHANGES PERMANENT AND REVERTING

When an editing change is made in a FRESS file, none of the altered pages are written back into the original file until another edit is done or the user explicitly accepts the change. This means that all relevant control information, as well as all altered pages, must be saved until the change is made permanent.

Each WCB, FUNAREA, and SCB is actually twice as large as the control block dsects show. Before an edit is done, MAINLINE saves each one in its second half. In addition, the stack indices in the UCB (DJRS, DMRR, DBTL) are saved in places provided in the UCB. Then if the user requests a revert, all of these are restored. This is done by RESTORE. Then RESTORE calls REVERT for each FCB. REVERT frees all the page slots attached to the FCB, adding them to the PAGEPOOL list from the UCB. It frees the spill table attached to the FCB, since all altered pages should be ignored. It decrements NUMUSED (the number of pages spilled to the spill file) by the number spilled from this file. When NUMUSED reaches zero, the spill file FCB is freed. (It would actually be possible to

free the spill file FCB immediately, since the purpose of the revert is to ignore all altered pages.) REVERT then reads in page zero and moves the reverted (unaltered) PCB to its place in the FCB.

When an editing change is to be made permanent, WRITEOUT is called. This routine writes all altered pages back to the original file(s). It loops through the FCB's, first checking for any pages dumped to the spill file. These are read into core and written back to the original file, and the spill table is freed. If all pages in the spill file have been switched back to the original file, the spill file FCB is freed. WRITEOUT then writes to disk any in-core altered pages attached to the FCB. The PCB is written out to page zero if it has been altered. Note that if WRITEOUT is terminated before the PCB has been written (e.g., if the system crashes) the file may be blown.

MAINLINE calls WRITEOUT to make an editing change permanent either when the user does an explicit accept or when another editing change is specified. The last edit is also made permanent when the file is freed (see next section).

2.10 FREEING A FILE

This occurs when either a free file (FF) or end (END) command is specified. PAGEEND is called to close the file. It calls WRITEOUT to write out all altered pages, returns all page slots to the free list from PAGEPOOL in the UCB, and does a LOGDISK to update the directory. When a free file is being done, FREEFCB is then called to freemain the core used by the FCB and rechain the FCB list around it.

2.11 DEBUGGING COMMANDS

There are a series of commands, most of them part of DBGMAIN, which are useful in checking and patching the contents of FRESS pages. Following is a description of each. Optional parms are in brackets. Many of the functions are directly parallel to paging functions. Display of a page is formatted as a 4 digit hex number indicating the displacement down the page of what is being displayed, followed by a number of 4-byte fields of the hex in that position on the page, followed by the character representation of the hex. The line

width and number of lines is determined by the normal display characteristics. Example:

```
0020  C1828340 C1C2C3C4 F1F2C1C2  *Abc ABCD12AB*
002C  F1F2F3F4 81828384 C1C2C3C4  *1234abcdABCD*
```

2.11.1 Debug File (DF)

DF filename

This opens a file without a password (i.e., if page 1 of the file is blown). The WCB is not initialized (DISPLAY will bomb if called) so use only debugging functions. Display is at the top of page zero (not the PCB).

2.11.2 Get Dictionary (GD)

GD decimal#

This gets the page corresponding to the dictionary entry given by decimal#. Display is at the top of the page.

2.11.3 Define Dictionary (DD)

DD decimal# hex#

This enters hex# (right justified) into the dictionary entry given by decimal#. Display is at the top of page hex#.

2.11.4 Get Page (GP)

GP hex#

This gets page hex# and displays at the top of the page.

2.11.5 Display Offset (DO)

DO hex#

This displays at the displacement hex# (rounded down to a double word) into the current page (i.e., the first page in the current PCB chain).

2.11.6 Find (FI)

FI [hex#] hexstring
(default of hex# is 0)

This searches the current page starting at hex# for hexstring. If found, display is at the nearest double word before the string.

2.11.7 Find Text (FT)

FT [hex#] characters
(default of hex# is 0)

This searches the current page starting at hex# for the character string characters. If found, display is at the nearest doubleword before the string.
Note: no translation is done on the character string, so lower case characters cannot be found this way on an upper case terminal.

2.11.8 Create Page (CR)

This gets the next free page, either from PCBFLIST or from a new block, and displays at the top.

2.11.9 Delete Page (RPG)

RPG hex#

This returns page hex# to the free chain from PCBFLIST.

2.11.10 Patch (PA)

PA hex# hexstring

This overlays the current page, starting at offset hex#, with the hexstring. Hexstring can be arbitrarily long. Any non-valid hex digits are translated to zero. Display is at the nearest doubleword before offset hex# into the current page.

2.11.11 Patch Text (PT)

PT hex# characters

This overlays the current page, starting at offset hex#, with the character string characters. Display is at nearest doubleword before offset hex#.

2.11.12 Multiple Patch (MP)

MP hex# decimal#

This overlays the current page, starting at offset hex#, with C'XX...XY', where the length of the string is decimal#. Display is at the nearest doubleword before offset hex#. This is an easy way to delete large sections of hex from main text, annotation, and work spaces. After the multiple patch has been done, the FRESS delete command can be used.

2.11.13 Page Links (PL)

This command causes a file to be created containing the name of each page in the current file along with the name of the next page, the previous page, and the DATAAREA and FREEAREA disps. The new file has the same name as the current FRESS file and a filetype of PLINKS. The PLINKS file may then be printed using the CMS Print or Offline Print commands.

2.11.14 Statistics (ST)

This command causes a file to be created containing the name of each page in the current file along with the number of bytes used on that page and the percentage of 1600 bytes this represents. Average statistics are also computed for each space. The new file has the same name as the current FRESS file and a filetype of STATS.

2.11.15 Pagin Statistics (PS)

This command prints at the terminal figures on the number of calls to SPLIT and PAGEBAL and the number of pages created and deleted. This information is kept in the UCB in the fields SPLITCNT, CREATCNT, PGBALCNT, and DELPGCNT.

2.11.16 Check (CK)

```
CK
CK ON
CK OFF
```

"CK" verifies dictionary entries, page links, and freearea and dataarea disps (where appropriate) in all spaces. "CK ON" does a "CK" and sets a bit in MODESWIT so that a "CK" is done after every edit command. "CK OFF" clears this bit. Messages indicate bad pagelinks (including dictionary entries) and bad freeareas with the pages involved.

This command is executed by CHECKFLE. The execution of the CK stops as soon as one error has been found, so another CK should be done after that problem is fixed to see if there are any more.

If no error is found, the system responds

FILE OKAY

The possible error messages and their meanings are:

BAD FREEAREA <page> <disp>>

This means the FREEAREA indicated at the top of the specified page does not point immediately after an x'6C07'. The <disp> given is the correct FREEAREA, assuming an x'6C07' appears on the page at all.

BAD PAGELINK <page1> <page2>

This means <page1> has an incorrect PREVPAGE in its HEADER. <page2> is the correct PREVPAGE based on the fact that <page2> pointed forward to <page1>. Thus if the forward link between <page2> and <page1> is incorrect, <page1> does not have an incorrect PREVPAGE. If either <page1> or <page2> is zero, it means one of the dictionary entries may be bad.

There is no longer a BAD DATAAREA message, since there is no longer a variable length pseudotext at the top of all pages.

2.11.17 Fix (FIX)

This is the panacea for almost whatever ails a file. It will reconstruct, upon request, the keyword space (k), internal label space (i), and structure space (s) of the file from the main text space. The procedure is to type 'FIX'. A read will be done to the terminal, and the letters of those spaces to be reconstructed should then be typed with no intervening blanks. When done, the response will be 'ALL FIXED'. On rare occasions, the problem is so bad that even FIX gets confused, and an error message will be produced instead. In this case, the problem which is confusing FIX must be fixed by hand before the rest of the problems can be solved.

This function should not be used indiscriminately. It is relatively expensive and can be avoided if there are only minor problems with a file. In any case, the problem should always be determined before FIX is used, since the point is to fix what caused the problem, not just to fix the file.

3 DATA STRUCTURE

3.1 STRUCTURE ORDERS

The 'data' in a FRESS file is made up of text strings, format codes, and structure (hypertext) orders. Format codes and text strings are treated the same, except when displaying or printing the file. Structure orders are different. They indicate some non-linear structuring of the file.

3.1.1 Format of Orders

The general format of an order is:

x'6C'<opcode><modifier><internal label>[<data fields>]

3.1.1.1 Opcode

The opcode of an order, which is 1 byte long, indicates what type of order it is. The opcodes currently defined in FRESS are

x'80'	point (location)
x'81'	tag
x'82'	block start
x'83'	block end
x'84'	table
x'85'	jump
x'86'	pmuj
x'07'	end of page

Note that except in the case of an end of page order, the high order bit of an opcode is always on.

3.1.1.2 Modifier

The modifier is also 1 byte long. The meanings or values of bits in the modifier byte are contained in the following table. For further explanation, see the discussion of the individual orders.

ORDER

BIT S/E	POINT TABLE	TAG JUMP/PMUJ	BLOCK		
0	1	1	1	1	-
1	0	0	0	0	-
0	0	annotation	annotation	attribute	-
0	0	area order	dec lab	value	s-
plice					
4	0	sketch ref	0	0	vie-
wspecs					
5	label	declab ref	label	label	-
0					
6	0	keyword	keyword	key ref	ke-
yword					
7	0	0	0	0	exp-
lainer					

NOTE: Bits 0-1 are unused in all modifier bytes
Bits 2-3 have meanings which need no data (except for table orders)
Bits 4-7 have meanings which indicate the presence of data fields

3.1.1.3 Internal Label

The internal label is a two-byte field which uniquely identifies the order. Internal labels are always negative and odd; they range from x'FFFF' to x'6D01'. This is to prevent an internal label from containing the digit x'6C' which would make it difficult to scan the data structure backwards as SCRLBACK does.

(See discussion of the internal label space for more information.)

3.1.1.4 Data Fields

The types of data fields which may be attached to orders are keywords, labels, viewspecs, explainers, and picture or decimal block references. The presence or absence of a particular data field is indicated by a bit in the

modifier byte. Label data fields must be 1 to 16 characters in length; keyword strings, viewspec strings, and explainers must be less than 256 characters. Each data field is followed by an x'00' indicating the end of the field. Label and viewspec data fields are displayed within parentheses; keywords within double quotes; decimal block and picture references within single quotes and blanks. Explainers are displayed with a trailing double percent sign. (This is a vestige of an obsolete design and implementation of jumps. See section on jumps.)

(See discussion of individual order types for more information on their data fields; also see discussions of the label and keyword spaces, and of the use of viewspecs in displaying a file.)

3.1.2 Points (Opcode=x'80')

A point, or location, order is used to "anchor" a label. (In the original implementation of FRESS, points were also used to anchor jumps, but this is not true anymore.) A point cannot exist without a label data field. Therefore the first three bytes of a point order are always x'6C8084' followed by the internal label of the point, the text of the label, and an x'00'. Points are displayed as %L(label). They (and the associated labels) are made using the Make Label (ML) command.

3.1.3 Tags (Opcode=x'81')

3.1.3.1 Area Orders

Area orders are tags with bit 3 (x'10') on in the modifier byte. Thus they appear in the data structure as x'6C8190<intlab>'. They are used to delimit areas in the main text, annotation, and work spaces. Areas are logically separate sections of a space; one cannot scroll into or out of an area. Area orders are displayed as *START OF TEXT AREA* and *END OF TEXT AREA* in the main text and work spaces, and as ***AREA*** in the annotation and structure spaces. (The structure

space does not actually have areas, it merely reflects the areas of the other spaces.) Note that the same area order can be a start or end area order depending upon whether it is viewed from the area before it or after it. Area orders are created by the Newarea (NA) and Splitarea (SPL) command. They can be deleted using the Delete (D) command; however, the first and last area lines in the main text space and first area line in the work space cannot be deleted.

3.1.3.2 Annotation Tags

Annotation tags are used to refer to a piece of text in a block in the annotation space. The internal label of the block start is 2 less than the internal label of the annotation tag. Because of this relationship, a piece of text in the annotation space will have a block start-block end pair around it for every tag which references it.

Annotation tags may have keyword strings associated with them. With the use of the Set Keyword Annotation Request String (SKA) command, annotations may be automatically shown in-line.

Annotation tags and their corresponding annotation blocks are made using the Insert Annotation (IA) and Make Annotation (MA) commands. A tag is made to an already existing block using the Refer To Annotation (RTA) command. Annotation tags are displayed as %T"keyword string" or %T.

3.1.3.3 Decimal Label Reference Tags

These tags are used to refer to a decimal block. Decimal blocks may appear in the main text, work, or annotation spaces, but they most often are only in the main text space. The data field of the tag contains the internal label of the block start to which it refers (followed by x'00').

They are created using the Make Decimal Reference (MDR) and Make Decimal Reference Deferred (MDRD) commands. They are displayed as %T'decimal#' where decimal# is the decimal number of the block referred to.

3.1.3.4 Picture Reference Tags

These tags are used to refer to drawings which exist in the picture space. The data field of the tag contains the picture number (negative, as internal labels - see discussion of picture space) followed by x'00'. They are created using the Make Picture Reference (MPR) command, and are displayed as %T'picture name'.

3.1.4 Block Start and Block End (Opcodes=x'82',x'83')

These orders are used to surround a piece of text which the user desires to consider a logical entity or block. The internal label of a block end is 2 less than its associated block start. If label or keyword data fields are associated with a block, they are attached only to the block start. Ordinary blocks (not annotation or decimal) are created using the Make Block (MB) command. All blocks are displayed as %< text in block %>.

3.1.4.1 Annotation Blocks

Annotation blocks appear in the annotation space and are referenced by tags from the main text or work spaces. If the keyword string attached to the tag is satisfied by the annotation keyword request string (DAKRSLOC in the UCB), the block will be displayed in-line directly after the tag which references it. When an annotation block is displayed in-line, the flow of text goes back to just after the annotation tag when the block end is reached. This was called instancing and was originally designed to apply to all splices to blocks (thus the use of an "instance" return stack.) See section on displaying a FRESS file for more detailed discussion. An annotation block has the same keywords associated with it as the tag which references it. These blocks are created using the Make Annotation (MA), Insert Annotation (IA), and Refer To Annotation (RTA) commands.

3.1.4.2 Decimal Label Blocks

A block which has the decimal label bit on in the modifier byte will be dynamically assigned a decimal number according to its position with respect to other decimal blocks when it is displayed online or printed offline. This feature is used to structure the text in a hierarchy. The decimal labels generated in this way can be used to retrieve the associated block using the Get Decimal Label (GDL) command.

Currently decimal label levels are correct in on-line display only within a single file; that is, sequential numbering will not continue if a splice or keyworded jump is taken to a new file. However, within a single file the numbering follows internal splices and keyworded jumps. In Fullprint, decimal label levels do follow interfile splices and jumps.

3.1.5 Table Orders (Opcode=x'84')

Table orders are entirely internal to the system and are used to keep track of and display keywords and labels. They appear only in the label display and keyword display spaces. See discussion of these spaces for more details.

3.1.6 Jumps and Pmujs (Opcodes=x'85',x'86')

Jumps and pmujs (reverse jumps) are used to link two associated pieces of text together. The internal label of a pmuj is 2 less than the internal label of its associated jump. If the splice bit (bit 3=x'10') is on in the modifier byte of a jump, it will be taken automatically as part of the flow of text while scrolling through or displaying the file. Unconditional jumps and pmujs are called splices and ecilpses. Keywords may be attached to the jump/pmuj pair. Then the Set Keyword Jump Request String (SKJ) command may be used to choose which jumps will be taken automatically. The same keyword string appears on both the jump and the pmuj. (Keywords may also be attached to a splice, although they will be ignored.

This is useful if the splice is later turned into a jump, since currently there is no way to add keywords to an order.)

Viewspec data fields may also be specified. These determine how the text on the other end of the jump or pmuj will be displayed (e.g., showing structure orders or format codes, how much formatting is to be done). Each jump and pmuj may have its own associated viewspec string.

A jump or pmuj may also have an associated explainer. This is a text string, presumably to explain what is at the other end of the link, with a maximum of 255 characters. It may contain text and format codes but no structure. When displayed, explainers are followed by a double percent sign.

In the original design and implementation of FRESS, jumps and pmujs did not have their own unique internal labels. Instead, a jump or pmuj had to be attached to a point or block order, and in the position which would normally contain a unique internal label, it had the internal label of the point or block on which the other end was anchored. A point or block could have more than one jump and/or pmuj attached, so one had jump "menus" which were simply a string of jumps and pmujs ended with a double percent sign. (This is the reason why explainers are now displayed followed by a double percent sign.) This design was changed in Summer, 1973 to make jumps and pmujs consistent with the format of other orders and to make implementation of the structure space more reasonable. Also, explainers were originally allowed to be of unlimited length and were not considered to be similar to other data fields. At the same time as the redesign of the jump and pmuj orders themselves, the maximum length for an explainer was set at 255 characters thus making it consistent with most other types of data fields.

3.2 SPACES

The pages in a file are divided into as many as 12 types: free pages, an initial page control block (PCB) page (page 0), protect pages (starting at page 1) and from 1 to 9 spaces of different types. Following is a

list of the different types of spaces and their space numbers.

1-Main Text
2-Annotation
3-Work
7-Structure
8-Label Display
9-Keyword Display
10-Internal Label
11-Dslink
12-Picture

3.2.1 Protect pages

The protect pages start at page 1. After the initial header information on the page, it contains one 40-byte entry for each password associated with a file. Each entry contains a 7-byte field for the text of the password, padded with blanks, 1 protect function byte, and 32 bytes of function mask. Each of the bits in these 32 bytes represents one of the possible 256 commands in FRESS (not all of which are currently used). If the bit is on, the function (or command) is allowed. All handling of the protect pages is done by the routine PROTECT. The function mask found by PROTECT is used in MAINLINE to determine if a particular function is allowed.

3.2.2 Main text space

This is the main text and hypertext space. It is delimited by area orders at the top and bottom, and possibly elsewhere if there are multiple areas. When the file is displayed online, the flow of text stops when an area line is reached going backwards or forwards. The main text space is created when the file is created and is never deleted unless the file itself is erased.

3.2.3 Annotation space

This space contains text which is considered annotation to text in the main text, work, or annotation spaces. It contains no area lines. A particular annotation will be surrounded by a block start-block end pair for each reference to it. If no references exist (i.e.,if the last reference to it is deleted) there will be no block start-block end pair. Annotations may contain both ordinary text and all kinds of hypertext. New entries are inserted at the end of the space. Entries are made using the Insert Annotation (IA) and Make Annotation (MA) commands. Text and hypertext may also be inserted directly, not as an annotation. A tag is made to an already existing block or piece of text in the annotation space using the Refer To Annotation (RTA) command. Selected annotation blocks may appear in-line by using the Set Keyword Annotation Request String (SKA) command. On a fullprint, these selected blocks will appear as footnotes. The annotation space is created when the first annotation is made and is deleted if the last annotation is deleted.

3.2.4 Work space

The work space holds text explicitly put there by the user to save it or work on it separately from the main text in the file. It is created when the first piece of text is moved or copied into it and deleted if the last piece of text in it is deleted or moved out unless a DSPTR still points inside it (which is usually the case). The commands used to manipulate the work space are Copy To/From Work (CTW,CFW) and Move To/From Work (MTW,MFW). The work space may contain all text and structure which can appear in the main text space. The work space has no end area order.

3.2.5 Structure space

The structure space contains a copy of all the structure in the file in its geographical order, that is, in the order in which it occurs in the file. (It

was originally called the decimal label space and contained only copies of the decimal block structure orders.) It reflects, in order, the main text, annotation, and work spaces. Unlike in the other spaces, all area orders are displayed as ***AREA***, but all other structure is displayed as it would be in the space being reflected.

Since the structure space contains structure in geographical order, any rearrangement of structure orders in the main text, annotation, and work spaces must appear also in the structure space. In addition, some editing on structure can be done in the structure space itself, which will then be reflected in the relevant space (see section on inverse editing). The structure space is used for scanning for counting of decimal label levels by DECLBSCN and for block nesting by MAKEBLOK.

The user may display the structure space to get an overall view of the structure of the file. In addition, by using the jump command and lp'ing any piece of structure in the space, the user may easily move to the corresponding position in the reflected space.

The structure space is created when the file is created and is never deleted. It is updated by any routine which makes a structure order, and by INSERT, DELET, KEYEDIT, and LABEDIT.

3.2.6 Label display space

This space contains all the labels in the file in table orders. They are arranged in alphabetical order. Editing a label can be done in the label space, the structure space, or at the associated point or block. The labels are displayed in columns, number of columns being dependent on the display line length. Each label table order is x'6C84' followed by the opcode and internal label of the order the label is actually atttached to, followed by the text of the label and x'00'. Entries are made in this space with the Make Label (ML) command, or by specifying a label field in a Make Block (MB) or Make Decimal Block (MDB) command. The space is created when the first label is created, and is deleted if the last and only

entry is deleted. It is updated by EDTLABEL and searched by FNDLABEL.

3.2.7 Keyword display space

This space contains an entry, in alphabetical order, for each keyword, attribute, and value in the file. The entry for an attribute is followed by entries for the values associated with that attribute, alphabetized among themselves. Each value is also listed in a separate entry inter-alphabetized with all other values, attributes, and keywords.

An entry begins with a table order. The modifier byte indicates whether the entry is for an attribute (x'20'), a value (x'10'), or an ordinary keyword. The order contains a unique internal label and the text of the keyword in a data field entry. The table order is followed by a list of references with which that keyword is associated. (However, the table order for an attribute is not followed by any references; instead, each associated value is followed by the references to that attribute value pair.) Each reference contains:

4 bits of flags: bit 0 (x'80') is always on
bit 2 (x'20') is on if reference is as a value
4 bit weight
opcode, modifier byte, and internal label of structure order to which keyword is attached (4 bytes)

For example, a keyword space which displayed as:

```
ATTR
  VAL   1 %<   1 %J   1 %P
KEY   2 %<                2 %T
VAL   1 %<   1 %J   1 %P
```

might appear internally as (explanations in parens):

```
6C84A0FFF7C1E3E3D900    (table order for ATTR)
6C8490FFF5E5C1D300      (table order for VAL as a
value)
A08282FFF1              (block reference for
ATTR:VAL,
                         weight 0)
```

```
A28582FFED              (jump ref for ATTR:VAL,
weight 2)
A28682FFEB              (pmuj ref for ATTR:VAL,
weight 2)
6C8480FFE3D2C5E800      (table order for KEY)
8082A2FFDF              (block ref for KEY)
8082A2FFE7              (block ref for KEY)
8081A2FFE1              (tag ref for KEY)
8081A2FFE9              (tag ref for KEY)
6C8480FFF3E5C1D300      (table order for VAL as
keyword)
A08282FFF1              (block ref for VAL)
A28582FFED              (jump ref for VAL)
A28682FFEB              (pmuj ref for VAL)
```

To facilitate keyword retrieval, there is a keyword space dictionary on page 0. It is the field PCBKYLAB in the FCB dsect. The dictionary consists of a halfword containing the maximum size of the dictionary, then a halfword containing the displacement from the start of the dictionary to the next free entry, then an entry for each page of the keyword space. Each of these entries contains the text of the first "valid" keyword entry on the page, padded with zeroes to 16 characters, and a 2-byte pagename. A "valid" entry is one which is alphabetized in the overall scheme of alphabetized attributes, values, and keywords. In particular, a value in its listing after its associated attribute would not be "valid"; a value listed alone in its alphabetical order with other values, attributes and keywords would be "valid". If no "valid" entry appears on a page, the keyword dictionary entry is a percent sign followed by 15 bytes of zeroes.

The keyword space is created when the first keyword is attached to a block, tag, or jump. It is deleted if the last keyword is deleted. It is updated by KEYWORDS. It is scanned by FINDKEY, and by KEYBUFF for displaying the space. It is also used in setting up block trails.

3.2.8 Internal label space

As mentioned previously, an internal label is a two-byte field which uniquely identifies the order with which it is associated. This space provides a means of finding a particular order given its

internal label. It contains a 3 byte entry (represented in the dsect ILABNTRY) for each internal label: a byte of flags and a 2-byte pagename of the page on which this label occurs in the text. (This is the page on which the structure order was created, not its reflection in the structure, label display, or keyword spaces.) The information in the first byte is:

Bit	Meaning
0	Free - always set to 1
1	Deleted order
2	Unused
3	Dslink space entry exists
4	Unused
5-7	Opcode of associated structure order

Currently no garbage collection is done on internal labels; that is, if a structure order is deleted, the internal label is not reused. There is a halfword field on page 0 (zero) containing the value of the next available internal label (PCBGUYID in the FCB dsect). The internal label space is updated by any editing routine which moves a piece of structure to a different page or creates a new structure order. It is created when the file is created and is never deleted.

The formula for finding the displacement from the first entry to the entry for a particular internal label is:

$$\frac{\text{FFFF - (intlab looking for)}}{2} \quad (* \quad 3)$$

If the displacement indicates that the entry is on the first page of the space (which is usually the case, since the first page contains entries for x'20C' internal labels), add to this the displacement to the first entry (currently x'1A') to find the displacement from the top of the page to the desired entry. If the entry appears on another page, additional manipulation is necessary. The length of a full page of internal label entries, not including the page header, is x'624'. If the displacement generated by the formula above is larger than this, x'624' should be successively subtracted until the displacement is less than x'624'. The number of subtractions is equal to the number of pages into the space the internal label lies. When the final figure less than x'624' is reached, this should be added to

x'1a' to determine where on the page thus determined the entry lies.

There is a dictionary for the internal label space on page zero (PCBINLAB in the FCB). It contains the page names (numbers) for each defined page in the internal label space. When it is determined how many pages into the space the desired internal label entry lies, this can be used to index into the dictionary to find the corresponding page name. (Note that there is space in the dictionary for only 30 of the possible 34 pages of internal label space. This seems to have been an oversight, but fixing it would require reassembly of many routines and it is unlikely that the limit will be reached anyway.)

3.2.9 Dslink space

This space maps interfile structure order references. For example, if an interfile jump is made from one file to another, the internal label of the pmuj should be 2 less than the internal label of the jump. The dslink space maps this expected internal label of the pmuj into the actual file name and internal label of the pmuj.

The space is 1 page long. It has a dslink code area at the top of the page and a dslink name area at the bottom. An entry in the name area contains the 8-byte CMS filename (padded with blanks) of the file which contains the desired structure order. The first entry is made at the bottom of the page, the next immediately before it, and so forth.

An entry in the code area is 6 bytes long and contains:

FILE - a 2-byte index (upward) to the relevant entry in the dslink name area
HERELAB - the (expected) internal label in this file
THERELAB - the internal label in the other file

All entries in the code area which refer to the same file will contain the same FILE field

As mentioned in the previous section, a flag bit will be set in an internal label space entry if an entry for that internal label exists in the dslink space. The dslink space is created when the first interfile jump is made. If an interfile jump is deleted, garbage collection is done so no unused gaps are left in the space. Unlike others, this space is not deleted if its last entry is deleted.

Notice that the current implementation of this space is entirely CMS dependent. Originally the entries in the name area contained a 6 byte volume name and a 44 byte dsname. This was changed (12/73) to increase the maximum number of interfile jumps. It is a relatively easy switch to make in the FRESS programs since only three routines - DSLINK, DELET (which update the space) and DSLNKSCN (which scans the space) - reference the space. However, if files are to be moved from one system to another, they must be converted.

Also note - the FILE index in the code area entry need be only 1 byte long. The additional byte was meant for now obsolete flags.

3.2.10 Picture space

The picture space of a file contains the figures created with the SKETCH function. The figures are stored as Imlac display orders. The first page of the space is a directory containing the name, number, and starting page of each picture. Each element in the directory contains 8 bytes for the name (padded with zeroes), followed by a halfword picture number and a halfword pagename. The picture number is always negative and never contains an x'6C' or x'00'. The numbers are assigned consecutively from x'FFFF', skipping those containing x'6C' or x'00'. Numbers assigned to pictures subsequently deleted are not reused.

The directory page is created when the first picture in a file is made and deleted if the last picture in a file is deleted.

Each picture begins on a new page. The header of the first page of every picture contains the length of the picture in IMLAC (16 bit) words

(PICTLEN). In the header of every page, the SKECHPAD bit in TEXTSCOP is set if the picture continues on the next page.

An internal FRESS routine, NANUPAGE, is used to get information from the directory. The routines which update the picture space are SKETCH, DPICT, SPICT, CPICT, and COPICT. The space is also accessed by NANUPAGE and LPICT.

The picture space was designed to allow the easiest possible integration with the rest of FRESS. It is therefore probably not the 'best' or most consistent design possible.

3.2.11 Unimplemented spaces

The space numbers 4, 5, and 6 were assigned to spaces which were never implemented. Following is a brief description of each. For further information, see older documentation.

3.2.11.1 Macro space(space 4)

This space was to hold text macros (not format macros, which do exist) which could then be inserted anywhere in the text of a file with the insert macro function.

3.2.11.2 Table of contents space(space 5)

Points were designed to have a bit in their modifier bytes indicating a Table of Contents point. An entry would then be made in this space in the correct geographical position. The entries were jumps attached to these 'table of contents points'; the associated explainers were the things printed. The space was meant to serve as both an online aid to viewing the file, and a means of printing a table of contents. This latter function has been implemented by the FULLPRINT programs.

3.2.11.3 Index space(space 6)

This space was to be similar to the T of C space, except that entries were to be alphabetical according to the explainers on the jumps rather than geographical. The string which was alphabetized was the explainer of a jump attached to the point order.

3.3 CONTROL BLOCKS

Following is a list of all the global FRESS control blocks. All of the control blocks are either MACROs or COPYs which exist in the FRESS macro library.

The control blocks are grouped loosely according to function. Each has a short description of its use and, where possible, which routine sets it up. More specific information will be found in sections on the routines which use the block. Check Table of Contents and Index.

NOTE: It is possible that the forms of these control blocks as listed here are not identical with those in the listings of some routines. In many cases, the routines were not affected by the changes made in the control block. If they were affected, patches were made to reflect this and the changes to be made in the source, if any, have been marked in the listings.

3.3.1 Storage Management

3.3.1.1 User Control Block (UCB)

The UCB is used to provide dynamic storage management for some of the other control blocks, for example, the window control block(s) (WCB), stack control block (SCB), and correlation map(s). It is also used to hold global status information easily accessible by all routines. It contains pointers or displacements to all other global control blocks. Throughout all of FRESS, register 7 points to the UCB.

The dynamic storage management for the UCB is done by GETUCB and FREEUCB which are called by any routine desiring to acquire or free some storage in the UCB. The static storage area and much of the dynamic storage initialization is done by IEAFRESS and MAINLINE. Many other routines update one or more of the static storage fields.

Following is a list of the static fields in the UCB with a short explanation of each. Where possible, the explanation is followed by the name of the routine or routines which set or update the field.

```
CURRWCB  DS   A      -> current WCB (MAINLINE)
USECOUNT DS   H      first byte=total number of
                       page slots
                       (IEAFRESS)
UCBLEN   DS   H      length of the UCB
                       (IEAFRESS)
TCBDCODE DS   X      device (terminal) code
                       (IEAFRESS)
TCBLNCNT DS   X      max number of lines in
                       buffer for
                       device
                       (MAINLINE)
TCBPTCNT DS   X      print count, overrides
                       TCBLNCNT
                       (MAINLINE,-
                       ATTEND)
TCBWCODE DS   X      window configuration
                       number
                       (MAINLINE)
SWPDCBAD DS   F      -> spill file info for
                       paging
                       (PAGING)
PAGEPOOL DS   0F     start of singly linked
                       list of
                       free page
                       slots
                       (IEAFRESS,
                       PAGING)
PAGECNT  DS   1C     number of free page slots
                       left
PAGEHEAD DS   AL3    -> first page slot free
                       (if any) -
                       IEAFRESS
SUBRLIST DS   A      -> PAGEROUT control block
ATTENPTR DS   A      -> low level attention
                       routine
```

```
                                        being used
                                        (dependent
                                        on device)
                                        (IEAFRESS)
FFLEAD   DS    A        unused; was -> first FLE
                                        which
                                        contained
                                        DSNAME
FCBPTR   DS    A        -> current FCB (GETFCB,
                                        FREEFCB,
                                        others)
SYSBLOW  DS    C        SYSBLOW+2 contains page
                                        size
                                        (IEAFRESS)
         DS    AL3      rest is unknown
SPLITCNT DS    H        # of calls to SPLIT
                                        (SPLIT)
CREATCNT DS    H        # of pages created
                                        (CREATE)
PGBALCNT DS    H        # of calls to PAGEBAL
                                        (PAGEBAL)
DELPGCNT DS    H        # of pages deleted (DELETE
                                        in PAGING,
                                        PAGEBAL,
                                        DELET)
GMESSLOC DS    H        disp to global message
                                        area
GMESSLEN DS    H        length of global msg if
                                        any
DVERFLGS DS    2X       mode flags (MAINLINE,
                                        ATTEND)
* first byte
VERFLGS1 EQU   x'80'    verify mode (obsolete)
VERFLGS2 EQU   x'40'    display mode
* note that ¬verify & ¬display => brief
VERFLGS3 EQU   x'20'    qualified verify mode and
                                        display top
                                        2 lines of
                                        buffer
VERFLGS4 EQU   x'10'    qualified display mode
VERFLGS5 EQU   x'08'    qualified brief mode
VERFLGS6 EQU   x'04'    seems to be unused (turned
                                        off in
                                        ATTEND)
VERFLGS7 EQU   x'02'    transcription mode
VERFLGS8 EQU   x'01'    qualified transcription
                                        mode
* second byte
VERFLGS9 EQU   x'80'    qualified static mode
TCBWNUMB DS    X        current number of windows
                                        (MAINLINE)
```

```
MSGFLAGS DS    X        miscellaneous flags
GPROMPT  EQU   x'80'    means  GMESSLOC contains a
                                prompt
SKNOLOAD EQU   x'40'    means  SKETCHPAD  does not
                                need
                                crossloadi-
                                ng to IMLAC
                                (MAINLINE,
                                SKETCH)
UNIT     DS    8C       display  info about device
                                (see
                                PLAYUNIT)
MODESWIT DS    CL1      other flags
FRSTCHAR EQU   x'20'    first  char  in buffer was
                                light
                                penned
JUSTIFYM EQU   x'10'    online       justification
                                desired
CKFFLAG  EQU   x'08'    check file (CHECKFLE) call
                                desired;
                                set  if  CK
                                ON      was
                                specified
NOUSRARE DS    CL1      last  window configuration
                                sent     to
                                non-360
                                side   (set
                                in
                                MAINLINE,
                                checked  in
                                ATTEND)
WCBLDISP DS    H        disp to WCB list which has
                                1 entry for
                                each
                                defined
                                WCB:
                                halfword
                                disp   from
                                top  of UCB
                                to     WCB,
                                halfword
                                length   of
                                WCB
WCBLEN   DS    H        # of entries in WCB list
FCBLDISP DS    H        was   disp   to  FCB  list
                                similar  to
                                WCB   list;
                                now  unused
                                since   FCB
                                list
                                replaced by
```

```
                                                  linked list
                                                  in
                                                  getmained
                                                  core   (not
                                                  GETUCB'd
                                                  core)
LASTGF    DS    H         file # of last file gotten
                                                  by  user gf
                                                  command;
                                                  set      by
                                                  MAINLINE,
                                                  used     by
                                                  DECLBSCN
SCBDISP   DS    H         disp   to   stack  control
                                                  block
SCBLEN    DS    H         length of SCB
FUNARDSP  DS    H         disp to function area
FUNARLEN  DS    H         length of function area
BUFFDSP   DS    H         used  for  calls to GETUCB
                                                  and
BUFFLEN   DS    H           FREEUCB for buffer space
COPYQDSP  DS    H         probably unused
COPYQLEN  DS    H         probably unused
FREAREAD  DS    H         disp to first free area in
                                                  dynamic
                                                  area     in
                                                  UCB; set by
                                                  IEAFRESS,
                                                  used     by
                                                  GETUCB/FRE-
                                                  EUCB
          DS    H         unused
SAVAREAD  DS    H         disp  to  savearea control
                                                  block
MERRCODE  DS    H         edit internal error codes
TRTTABLE  DS    CL256     system      trt     table,
                                                  containing
                                                  stops   for
                                                  x'00'   and
                                                  x'6c'.   If
                                                  changed  in
                                                  a  routine,
                                                  must     be
                                                  restored
                                                  before
                                                  returning
                                                  or  calling
                                                  any   other
                                                  routine.
OSDECB    DS    7F        probably  unused  (in  CMS
                                                  version)
```

```
IOPTR    DS    A       probably unused (in CMS
                                 VERSION)
BBNXTFNM DS    H       next file # available for
                                 GETFCB
DISPWCB  DS    H       disp to WCB passed to
                                 DISPLAY
                                 from
                                 MAINLINE
UCBSVDSP DS    H       disp to paging save area
                                 control
                                 block
                                 (stack of
                                 saveareas
                                 used by
                                 paging
                                 routines)
UCBSVLEN DS    H       length of save area
                                 control
                                 block
UCBFSCBP DS    F       ptr to FSCB for this UCB
DSTRINGS DS    0H      disp to and lengths of
                                 various
                                 text
                                 strings
DSCANLOC DS    H       disp to last scan string
                                 (from last
                                 locate
                                 command)
DSCANLEN DS    H       length of same
DBTRSLOC DS    H       disp to block trail
                                 request
                                 string
                                 (BTC,BTD)
DBTRSLEN DS    H       length of same
DJKRSLOC DS    H       disp to jump keyword
                                 request
                                 string
                                 (SKJ)
DJKRSLEN DS    H       length of same
DAKRSLOC DS    H       disp to annotation keyword
                                 request
                                 string
                                 (SKA)
DAKRSLEN DS    H       length of same
DBTL     DS    C       index to top entry of
                                 block trail
                                 stack
DMRR     DS    C       index to current ring
                                 stack entry
```

DJRS	DS	C	index to current jump return stack entry
FCBLIST	DS	A	->start of list of FCB's of open files
SDBTL	DS	C	to save DBTL for possible revert
SDMRR	DS	C	to save DMRR for possible revert
SDJRS	DS	C	to save DJRS for possible revert
STACKTOP	DS	9F	stack for choosing default window for user command (MAINLINE)

3.3.1.2 Free Storage Control Block (IEUFSCB)

This control block holds pointers to the dynamic storage getmained by various routines. It is managed by IEAFARTS, which is called by these routines when they need temporary storage.

3.3.2 File handling and paging

3.3.2.1 File Control Block (FCB) and Page Control Block (PCB)

These two control blocks are combined together. The PCB is a section of page 0 (zero) of the file which contains paging control information. When the storage for the FCB for a file is allocated, these portions of page 0 are copied into the bottom of the FCB. The FCB contains global information about the file such as the password currently in effect, the file number, etc. All the FCB's are linked together in a doubly linked list originating in the UCB (FCBLIST).

3.3.2.2 Page Header (HEADER)

This is the format of the control information at the top of each page.

3.3.2.3 Paging Routines (PAGEROUT)

This is a table containing the addresses of the paging routines which appears at the top of the PAGING csect and is pointed to be SUBRLIST in the UCB. Other routines use the entries in the table for calling the paging routines rather than having V-type constants. This was done because of restrictions when running FRESS under MVT. (Many of the paging routines are actually entry points within a load module, and thus cannot be addressed by V-cons within another load module under MVT.)

3.3.3 Data Structure

3.3.3.1 ORDER

This contains the general format of a structure order and EQU's for all the modifier byte meanings for the various orders. It is used throughout the system when a routine looks at an order in the data structure.

3.3.3.2 ILABNTRY

This is the format for an internal label space entry.

3.3.4 Command language interpretation

3.3.4.1 Data structure pointer (DSPTR)

The DSPTR contains all the information about a hit point (in the data structure) which is necessary to an editing routine. It is set up by CLINTERP: CLINTERP calls SCANNER to find the location in the buffer of a text string from the command line; SCANNER calls CORMAP to resolve this

buffer location into a page and displacement. One DSPTR is returned for a hit and two DSPTRs are returned for a scope. 'intelligent terminal' hits are directly resolved into DSPTRs.

3.3.4.2 Function Area (FUNAREA)

The function area contains information about the status of the system with respect to user commands, e.g., the current function element, a stack of function elements to be executed, and any incomplete functions. It is set up by ATTEND and updated by ATTEND, CLINTERP, and the low level attention routines.

3.3.4.3 Function Element (FUNCELEM)

This contains all the relevant information about a single user command. It is formed by CLINTERP from the command line typed by the user. It is used by MAINLINE and by any routine called by MAINLINE (with the exception of display routines) to handle that function.

3.3.4.4 Function Table (FCNTABLE)

This is actually four related tables containing information about each function and its parameters. It is used by CLINTERP to recognize commands in the command line and match parameters.

3.3.5 Displaying a file

3.3.5.1 Window Control Block(WCB)

This is the major control block used by the display routines. It contains information about a window, including size, what should be displayed in the window, and displacements to the associated correlation map and buffer. It is set up by MAINLINE and updated by display routines. Various other routines update some of the flags. There is a WCB for each window defined in the system.

3.3.5.2 Viewing Specifications (VIEWSPEC)

This is the format of a 7 byte string specifying how the file should be displayed. It controls things such as whether particular structure orders are to be displayed, whether format codes should be displayed, and how much on-line formatting should be done. A viewspec string is formed by VIEWTRAN when called by VIEWMAIN (after a set viewspecs command), ORDERS (when a viewspec string appears on a jump in a scroll or while filling the buffer), or TAKEJUMP (when a viewspec appears on a jump or pmuj light-penned in a jump command).

3.3.5.3 Correlation Map entries (CORRMAP1 and CORRMAP2)

The correlation map correlates a buffer position to a place in the data structure. It is formed by calls to MAKEMAP1 and MAKEMAP2 (entry points in MAPROUT) from other display routines. CORRMAP1 is the form of a first level correlation map entry, which is made when a new page of the data structure is used in filling the buffer. CORRMAP2 is the form of a second level correlation map entry, which contains a file, page, and displacement into the data structure, and the buffer displacement which corresponds to the same piece of text. (See section on display routines for more detailed explanation of correlation maps.)

3.3.5.4 Stack Control Block (SCB)

The SCB is a piece of storage used to allocate entries for the various stacks (see succeeding entries). The static entries at the top of the SCB contain information about the current storage allocation.

3.3.5.5 Jump Return Stack (JRS)

The JRS entries are pointers into the data structure indicating a position the system wishes to remember (save) for back-tracking along a user's path in a session. Entries are made by MAKEJRS on calls from MAINLINE (for system-defined

saves) and from SAVE (for user-defined saves). They are updated by UPDTDSAR, if necessary, when editing is done.

3.3.5.6 Memory Return Ring (MRR)

In addition to a linear trace of one's path through a file, the system can also save a specified set of positions in a ring. These would be entries in the MRR, each of which is a pointer into the data structure. The entries are made by MRRMAIN, and are updated by UPDTDSAR, if necessary, when editing is done.

3.3.5.7 Block Trail (BTL)

A block trail is a list of keyworded blocks to be displayed in sequence. Unlike the other stacks, it is built all at once rather than one entry at a time. Also, the entries contain the file number and internal label of the block rather than a pointer into the data structure, so they need not be updated when editing is done. The entries are made by MAKEBTL and are used by ORDERS and TRAIL.

3.3.5.8 Splice Return Stack (SRS)

This stack is used so that a splice or keyworded jump taken when scrolling forward through a file or group of files will be taken in the other direction when scrolling backwards. The entries contain the file numbers and internal labels of both ends of the jump or splice. The entries are made in ORDERS and are used in SCRLBACK.

3.3.5.9 Instance Return Stack (IRS)

This stack is used to keep track of tags to annotation blocks which are taken automatically in-line. It is known as instancing because when the block end is reached, display returns to the place in the data structure immediately following the tag. Thus the same 'master' annotation block can have more than one 'instance' in the file. Each entry in the IRS contains a file number and

internal label of both ends (the tag and the block start. This is not really necessary since the internal label of the block start is 2 less than that of the tag, and they are in the same file. However, the IRS was originally meant to take care of all splices to blocks.) The entries are made and used by ORDERS.

3.3.5.10 PLAYUNIT

This is the format of an 8-byte entry in the UCB called UNIT. It gives information about how to set up display lines for the particular terminal being used, for example, whether carriage control is needed before and/or after a line, and if so, what the carriage control character is. It is used by NEWLINE, ENDLINE, and QUERY. DBGMAIN checks the flags in UNIT but has them EQU'd inside the routine rather than using PLAYUNIT.

3.3.5.11 SCOPE

The SCOPE block is used by both display and editing routines. The editing routines use it to find out information about a hit point. Display routines use it to find out information about a position in the data structure to be used in filling up the buffer. The upper part of the block is filled in by BLOCSCAN. The lower part (which appears in the macro expansion by specifying ROUT=DISPLAY) is filled in by DECLBSCN and updated by assorted display routines. The actual SCOPE storage is in the WCB for each window.

4 ATTENTION HANDLING

This section discusses what happens when FRESS requests a command from the user. This is probably the most complex process in the system. The sequence is as follows:

1. MAINLINE determines a command is needed and calls ATTEND.

2. ATTEND initializes and cleans the function area (see below) and determines if any messages need be printed. It does a lot of flag shuffling, and no one really knows everything it does. ATTEND then calls a low-level attention routine to get a command line.

3. The particular low-level attention routine is determined by the device being used as a terminal. The four low-level attention routines in existence are ATTNCMS (for DATELS, 2741's, TTY's, etc. under CMS), ATTNBUGS (for Vector General), ATTNIMLC (for the IMLAC), and ATTN2260 (for 2260 console in Holland). The low level attention routine does the actual reading from and writing to the terminal, including messages and buffer printing. When a command is read from the terminal, low-level attention checks to see if the first command on the line is one of the house functions which it handles. If not, it returns the command to ATTEND.

4. ATTEND checks for a few special cases, and then calls CLINTERP to parse the command line.

5. Using FCNTABLE, CLINTERP creates a function element in the function area for each command on the line, calling SCANNER to resolve any hit points. It then returns to ATTEND.

6. ATTEND returns to MAINLINE, passing back one function element.

7. MAINLINE determines which routine to call to handle the command returned by ATTEND. In a few cases (e.g. HELP, SW, CW) MAINLINE handles the command itself.

4.1 FUNCTION AREA AND FUNCTION ELEMENTS

The function area (see the associated diagram) contains a stack of function elements; a stack of data structure pointer parameters, the fixed pointer stack; a literal pool for literal parameters; and a stack of data structure pointers, the level pointer stack, which are returned by functions and are used as implied insert points. Each function element contains the opcode and a byte of flags followed by a list of pointers. The pointers are displacements from the UCB to the parameters. The pointer can be a displacement to a DSPTR in the fixed pointer stack or a complemented displacement to a literal which is stored as a halfword aligned length followed by characters in the literal pool. If the parameter is an unspecified optional parameter, the pointer is zero . The list of pointers always ends with an X'FF' and a byte containing the number of times to pop the level pointer stack.

All displacements are from the UCB.

4.1.1 FUNAREA

```
FUNNOW   DS    H     disp to the current function
element
FUNWAIT  DS    H     disp to a waiting function
element
FUNSAVE  DS    H     disp to top of function element
stack
FUNNULL              disp to null function element
LEVBOUND DS    H     and upper bound of level pointer
stack
FUNNEXT
PTRRET   DS    H     disp to returned dsptr
NESTDPTR
LEVELPTR DS    H     disp to top of level pointer
stack
FIXEDPTR DS    H     disp to top of fixed pointer
stack
FUNVER   DS    H     disp to verify dsptr (obsolete)
FUNSVFLG DS    X     flags of funnow are saved here
                     (on return to ATTEND)
FUNFLAGS DS    X     control flags in function area
FFLAG1   EQU   X'80' a ptr has been returned
FFLAG2   EQU   X'40' have an input line for CLINTERP
                     from REJECT/ACCEPT processes
```

```
FFLAG3   EQU   X'20' an error occurred (in
EDITMAIN,MAINLINE, etc)
FFLAG4   EQU   X'10' unused
FFLAG5   EQU   8     suppress display until next GET
FFLAG6   EQU   4     "WARNING" count in R2, not an
error
                     for the question mark convention
FFLAG7   EQU   2     set with FFLAG5 means do not
suppress
                     the last buffer or verify
generated
ISFLAG   EQU   1
FUNWAIT1 DS    H     secondary waiting function
FUNCRPCT DS    H     repeat count
FUNOUTF  DS    X     flags for output to user
                     set for low level attention
FUNOUTF1 EQU   X'80' GMESSLOC    points   to   global
                                 message     to    be
                                 printed
FUNOUTF2 EQU   X'40' message from ATTEND to be
printed
FUNOUTF3 EQU   X'20' the buffer is to be printed
                     (from third line onward)
FUNOUTF4 EQU   X'10' the message lines are to be
                     printed after the buffer
FUNOUTF5 EQU   8     get input
FUNOUTF6 EQU   4     R2 is displacement to ? Error
FUNOUTF7 EQU   2     if  set, going  into input mode;
                               all   but   ATTNIMLC
                               type INPUT
FUNOUTF8 EQU   1     used  by  ATTNIMLC; means window
                               data being sent
FIXBOUND DS    H     disp to bottom of the fixed ptr
                     stack and level ptr stack
FUNCPNUM DS    H     number of lines to be printed
LITERALS DS    H     disp to the literal pool
FUNCLINT DS    F     CLINTERP stuff
USERTPTR EQU   FUNCLINT address of user FCNTABLE
(lower 3 bytes)
MODEFLAG EQU   FUNCLINT modeflags set by CLINTERP
ASISMODE EQU   X'80' as-is mode
SUPRMODE EQU   X'40' suppressed display mode
USERFTAB EQU   X'20' user function table exists
IMPLFLAG EQU   X'10' pointer has been returned
INPTMODE EQU   x'08' INPUT MODE is in effect
DELIST               dynamic character values
DLSTRING DC    C'$'
DLPOINT  DC    C'|'  line character displacement
delimiter
DLGO     DC    C'>'  command separator
```

```
DLZZZ    DC    X'15' carriage return - end of the
line
DLKILL   DC    C'<'  do not regen display for one
command
DLNOVIEW DC    C'('  no display until DLVIEW
DLVIEW   DC    C')'  display
DLUPLEV  DC    C'|'
DLATTN   DC    C'&'
DLFILL   DC    C'?'  fill in incomplete function
DLHYPER  DC    C'%'  hypertext function
DLMINUS  DC    C'/'
DLDEF    DC    C'?'  deferred lp hit
DLREF    DC    C'='  lp back reference
DLTEXTBL DC    C'_'  text blank
DLSPOT   DC    C'&'  hit point in pattern
DLELIP   DC    C'.'  ellipsis character
DLINTER  DC    C'/'
DLQUAL   DC    C'-'  command qualifier delimiter
         DC    X'00'
FUNSCAN  DS    12C   verify dsptr (obsolete)
FUNCOML  DS    F     pointer to work area
FUNCOML1 DS    F     again count and ptr to work area
         DS    H     area before function elements
FFELEM   DS    H     start of function elements
*FOLLOWING FLAGS ARE USED BY CLINTERP
PTRPAIRF EQU   X'80' paired dsptr flag
ENDFCND1 EQU   X'FF' end of function element
ENDFCND2 EQU   X'00'
IMPHITFG EQU   1     implied insert point parm
DEFHITFG EQU   3     deferred hit flag
DEFSCPFG EQU   5     deferred scope flag
FREELIT  EQU   X'80' literal to be freed
```

4.1.2 FUNCELEM

```
FUNCOPCD DS    X     opcode of function
FUNCFLAG DS    X     flags for function element
TEXTHYPR EQU   X'80' set if hypertext function
WAITFLAG EQU   X'40' set if waiting for completion
                    of other functions
DIS4FLAG EQU   X'20'
UNSENCPT EQU   X'20' ACCEPT needed for unseen text
REPTFLAG EQU   X'20'
HYPRTCPT EQU   X'10' ACCEPT needed for hypertext
QUALFLAG EQU   X'10'
EDTERROR EQU   8     error  found,  error  msg  to be
                              printed.  Also print
```

```
                                        ACCEPT OR REJECT msg
NPARFLAG EQU   8
DIS1FLAG EQU   8
RETPTRS  EQU   4     pointer to be returned by edit
RET2PTRS EQU   2     another pointer to be returned
by edit
REGDFLAG EQU   1
REGNDSPL EQU   1     regenerate the display
ATTFLAGS EQU   TEXTHYPER+RETPTRS+RET2PTRS+REGDFLAG
PTR1     DS    H     first parameter of function
         .
         .
         .
PTR8     DS    H     last parameter of function
```

4.2 ATTEND

Function - ATTEND gets a command for MAINLINE.

Entry points - ATTEND

Entry conditions - none

Exit conditions - FUNNOW in FUNAREA is a displacement from the UCB to the function element (FUNCELEM) in FUNAREA to be executed.

What it changes - FUNAREA

Who it calls - low level attention handlers, CLINTERP, and REVERT

Who calls it - MAINLINE

Error messages - none

Storage used - 72 bytes via IEAFARTS and a 512 byte

Registers used -
- R5 points to the command line
- R6 points to FUNAREA
- R7 points to the UCB
- R9 points to the WCB

CMS dependencies - 512 byte buffer is GETMAINed

Important flags - FUNOUTF, FUNSVFLG, FUNFLAGS, MODEFLAG in FUNAREA; FUNCFLAG in FUNCELEM (see function area documentation).
MSGFLAGS, DVERFLAGS in UCB; DFLAGS in WCB.

Algorithm

Note that names in parentheses are labels in the program where indicated action is taken.

ATTEND calls a low level attention routine for a command line (FORINP1), then calls CLINTERP. CLINTERP parses the command line and creates a function element in the function area for each command. ATTEND passes the function elements to MAINLINE one at a time (NEXT).

ATTEND also initializes the function area (FUNINIT0) and cleans the function area (NOWRE1). It handles messages for HYPERTEXT functions (ATTEN1 and GETACC) and waiting functions (FILL1). It maintains a stack of DSPTRs, the level pointer stack. DSPTRs returned by a function are added to the stack (ATTEN2). The DSPTR at the top of the stack is filled into any function element which needs the implied insert point (FILLRET). The stack is popped by the &P and &L house functions.

ATTEND also handles the TYPE function (PRINTMNY).

4.3 FUNCTION TABLE

4.3.1 Creating and updating the function table

The function table used by CLINTERP is generated by a series of macros, one per function, in the file FCNTABLE. The macros are in the MACLIB FRESS. The basic program is:

```
FCBTABLE CSECT
         FCNTAB
         FCN <parameters>
         FCN <parameters>
         .
         .
         .
         ATABLE
         BTABLE
```

```
          CTABLE
          END
```

The macro FCN, which uses submacros FB and PF, generate entries in the following tables:

ATABLE: 26 halfword entries, indexed by the first letter of the function name. Each entry is a displacement to the BTABLE entry for the first function starting with the corresponding letter.

BTABLE: one entry per function. Each entry has
1 byte for function name length
1 byte for function opcode
n bytes for the function name in characters.

CTABLE: one halfword entry per function, indexed by twice the opcode. Each entry is a displacement to the corresponding DTABLE entry.

DTABLE: one entry per function. Each entry has
1 byte of function flags
2 bytes of flags for each parameter
1 byte length of the DSPTR area.

Note: all displacements are from the start of the CSECT FCNTABLE.

4.3.2 Filling in the FCN macro

The basic format of the macro is

```
 FCN
<name>,<opcode>,<funcflags>,<parm1flags>,...,<parmnf-
lags>
```

where

<name> is the function's name (it must be less than 16 characters);
<opcode> is the opcode in decimal of the function;

<func flags> are the functions flags, described below;
<parm flags> are the parameter flags, described below. The parameters should be in the order expected by the function routine.
If more than one flag is specified, they should be enclosed in parentheses and separated by commas. The flags may be in any order.

Function flags:
HT - this is a hypertext function
Q - qualifiers are allowed
R - the function is repeatable
P1 - the function returns 1 pointer
P2 - the function returns 2 pointers
RD - the display should be regenerated after completion
X - none of the above

Parameter flags:
<parm flags> ::= (<keyword>,<semantic type>,<opt>) where
<keyword> is the keyword to be associated with this parameter. The keywords are: AT, TO, FROM, DATA, OPT, KEY, and FOR. Every parameter must have a unique keyword associated with it. Keywords were once used by CLINTERP to recognize parameters.
<semantic type> is one of those described below
<opt> is optional and is O if the parameter is optional.
The semantic types are:
HIT - parameter may be a pen hit
SCP - parameter may be a scope (1 or 2 hits)
IMP - parameter may be an implied hit (O and HIT)
NUMB - parameter is a number with 1 to 3 digits
SNUM - parameter is a number with 1 to 4 digis including a sign
PASS - parameter has up to 8 characters and is a password
LABEL - parameter has up to 16 characters and is a label
FILE - parameter has up to 53 characters and is a filename

TEXT - parameter is a literal of arbitrary length, and can be preceded by an as-is code
LIT - parameter is a literal of arbitrary length, and cannot be preceded by an as-is code
RHIT - dummy parameter, space is left for a hit
RSCP - dummy parameter, space is left for a scope
WINDO - window number parameter, conditional NUMB, that is ignored for single window terminals
FILL - used internally by CLINTERP to fill in an incomplete function

Some examples of function table entries -
FCN COPY,113,(Q,RD),(FROM,SCP),(TO,HIT)
FCN MFILE,5,(RD,P1),(FOR,FILE),(AT,WINDO,O),(KEY,PASS,O)

4.4 CLINTERP

4.4.1 Overview

Function - CLINTERP parses a command line and creates function elements for each command in the function area.

Entry Point - CLINTERP

Parameters - There are no parameters. On entry Register 5 points to the beginning of the command line.

Return Codes - If an error occurred, R15 = 256 * (the displacement to the error in the command line) + error code. Otherwise R15 = 0.

What it changes - The command line, the function area (FUNAREA), and function elements (FUNCELEM).

Who it calls - SCANNER, GCOLLECT, and ERRMESS. GCOLLECT and ERRMESS are entry points in CLI2SECT.

Who calls it - ATTEND

Error messages -

1 &G count or &P count or line displacement or character displacement greater than 127.
2 Unknown command name
3 Invalid hex code in the command line
4 Too few parameters given
5 Error in a deferred scope
6 Function takes no parameters
8 Unsupported command
9 Invalid or illegal qualifier
10 Not enough space in FUNAREA
11 Incorrect value in parm position
12 Abend - programming error
13 Second incomplete function encountered
15 Pattern not found by SCANNER
16 Invalid pattern
17 Tells ATTEND to clear the level pointer stack in function area
18 Tried to pop top of level pointer stack in function area
19 Stack full - Obsolete
20 No plist found - Obsolete
22 Tried to repeat non-repeatable function
23 Tried to fill nonexistent incomplete function
24 Length error, scroll or number parameter too large
25 Unlightpennable text
26 No function to be repeated
27 Invalid separator after function

Storage used - X'214' bytes via IEAFARTS

Registers used -

R5 LINEPTR current position in command line
R6 FUNPTR pointer to FUNAREA
R7 UCBPTR pointer to the UCB
R8 LINELIST pointer to current LLSECT in the line list (see parameter parsing algorithm in next section)
R9 FCNLIST pointer to current function's DTABLE entry
R10 FCNPLACE pointer to current FUNCELEM in FUNAREA
R11 and R12 are base registers
R13 points to the data area

CMS dependencies - none. CLINTERP is too large to assemble on a 384K machine, but it will assemble in 832K.

```
     Important flags -
FUNFLAGS   and  MODEFLAG   in  FUNAREA,  FUNCFLAG  in
FUNCELEM (see Function Area documentation)

DTABLE entries in FCNTABLE, PARTBYTE and PARSBYTE

PARTBYTE - parameter type byte
OPTPARM  EQU   X'80' optional
LITOKFLG EQU   X'40' literal
HITOKFLG EQU   X'20' hit
SCPOKFLG EQU   X'10' scope
LASTPARM EQU   X'08' last parameter in the list

PARSBYTE - paramter semantics byte
IMPOKFLG EQU   X'80' implied insert point
RSCPFLAG EQU   X'10' dummy scope
RHITFLAG EQU   X'08' dummy hit
FILLFLAG EQU   X'20' fill in incomplete function
SEM1CK   EQU   X'07' semantic type field
SUMFLAG  EQU   X'05' semantic type number
FILEFLAG EQU   X'02' semantic type filename
TEXTFLAG EQU   X'01' semantic type text

ERRWARN - warning flags in CLINTERP
LENBLOW  EQU   X'80' length check in semantics
FLITFLAG EQU   X'40' freed literal overlayed
BADCHAR  EQU   X'20' invalid character translated to
blank
ANDBOMB  EQU   X'10' set by &B for debugging under
MVT

PREFLAGS preliminary function flags in CLINTERP
         <same flags as FUNCFLAG in FUNCELEM>

LLTYPE in LLSECT
LL2HITS  EQU   X'80' deferred scope or two intell
term hits
LITOKFLG EQU   X'40' literal parameter
HITOKFLG EQU   X'20' hit
SCPOKFLG EQU   X'10' scope
LLDEFR1  EQU   X'08' first hit of scope deferred
LLDEFR2  EQU   X'04' second hit of scope deferred
LLINTRM1 EQU   X'02' one intell term data
LLINTRM2 EQU   X'01' second intell term data
```

4.4.2 Algorithm

Note that names in parentheses are labels in the program where the indicated action takes place or the name of a storage location.

After initialization, the command line is scanned forward (PRIME) to check for illegal characters and for a carriage return. If no carriage return is found, an X'FF' is stored at the end of the command line. Then the line is scanned backward to check for any dot convention (WEDGEA).

The main loop of CLINTERP goes forward through the command line, analyzes each function and its parameters, and creates a function element (FUNCELEM) in the function area (FUNAREA). If the input mode flag is set (INPTMODE in MODEFLAG), then the rest of the line is not checked for functions but saved as input (GETINPUT). A translate table is set up by a subroutine (DELRESET) to interpret the first character of a function (START).

Blank and '>' (DLGO) are skipped (FISTART).
'(' (DLNOVIEW) sets flags that suppress the display until another function starts with ')' (FSUPDL).
')' (DLVIEW) clears flags that suppress the display (FSUPDE).
'<' (DLKILL) sets flags to suppress the display for a function (FUNAPDF).
'%' (DLHYPER) sets flags for Hypertext function (FHYPER).
'_' indicates enter input mode (FIMPIN).
X'55' indicates intelligent terminal data (FINTCD).
'?' (DLFILL) indicates filling of an incomplete function (FINCFL).
A to Z and a to z are the first letters of a normal FRESS command (FCOMNM).
'&' indicates a House function (FHOUSE).
'+', '-', or a number indicate a scroll (FSCROLL).
X'FF' or X'15', carriage return, marks the end of the command line (RETURN).
any other character is an error (UNKNERR).

For a scroll (FSCROLL), a function element is created. The number is stored in the function area as a literal parameter.

House functions (FHOUSE) are mainly for setting modes so they are performed by either CLINTERP or ATTEND or low level attention routines. For the &P House function, CLINTERP puts the number of levels to pop in the byte following the X'ff' that marks the end of the function elements in the function area.

If the first character indicates a normal FRESS command (FCOMNM), a subroutine (GETFNAME) is called to try to find the command name in FCNTABLE. If the command is found, its DTABLE address and internal code from BTABLE is stored in FCNSAVE. Then the next character in the command line is checked for a qualifier (CKQUALB). '-' (DLQUAL) is the qualifier delimiter.

Next the parameters are parsed (GETLINE). The DTABLE entry in FCNTABLE of each function is searched and the number of required parameters and the maximum number of parameters is counted (GETL1). The next character in the command line (after the command name and qualifier, if they exist) is set as the key delimiter (GETL1E). For each parameter a linelist element (LLSECT) is created to describe the parameter. It contains a displacement (LLPAT) to the parameter from the beginning of the command line. The length of the parameter is stored over the preceeding delimeter byte in the command line. A null linelist is left for each dummy parameter (RHIT or RSCOP). The end of the parameter parsing (GETLCR) is indicated by a carriage return or a non-literal command separater '>' (DLGO).

The linelist elements are matched with the parameter specifications in DTABLE. If the number of linelist elements is less than the number of required parameters (GETLCR2), an error is returned (TOOFEW) unless the function is an insert function, i.e. INSERT or SUBSTITUTE. If the number of linelist elements is greater than or equal to the number of required parameters but less than the maximum number of parameters (FIXOPT), then starting from the last parameter specification in DTABLE (PARTBYTE and PARSBYTE) and the last linelist element, each PARTBYTE is tested until an optional parameter is reached. The linelist elements of the required parameters are moved down to match the required parameters at the end of the list of parameters in DTABLE. The vacated linelist elements are zeroed. Thus required parameters are filled in first, then optional parameters are filled in from left to right.

Note: When specifying the parameter list for a function, all the optional parameters must be consecutive. If R is any number of required parameters and O is any number of optional parameters,{then legal parameter list specifications are R, O, RO, OR, or ROR. Any other combination would not be parsed properly, i.e. ORO would not work.

Also note: An implied insert point (IMP) parameter is defined as optional. An IMP must be the first parameter in a parameter list since ATTEND checks only the first parameter for filling in an implied insert pointer. The literal parameter that specifies input for an insert function must be the last parameter since its length is checked for zero for enter input mode.

Now that the parameters are matched, a function element can be created. If the function save area is full (CGCOLFCN), an external routine (GCOLLECT) is called to clear space. Then the opcode (FUNCOPCD) and function flag (FUNCFLAG) are stored in the function element (FNROOMOK). Next each parameter is analyzed according to the specification in DTABLE (PARTBYTE and PARSBYTE) and the proper bits for a scope, hit, literal, or deferred parameter are set in the linelist element (LLTYPE). '?' (DLFILL) indicates a deferred LP. For literals (ANALLIT), the length of the character string is checked. If the literal is a number, it is checked for valid digits. For hits (ANALHIT) and scopes, the parameter is checked for intelligent terminal data, deferred LP, or a leading line and character displacement. '|' (DLPOINT) is the displacement delimiter.

Then each parameter is analyzed again according to the flags set in the linelist (LLTYPE) and finally added to the function element. A literal (PARLIT) is added to the literal pool in FUNAREA and the displacement from the UCB to the literal is stored in the parameter list. If the literal pool is full, an external routine (GCOLLECT) is called to clear space. A hit (PARHIT) is resolved into a data structure pointer (DSPTR) by calling SCANNER (PUTINHIT). The DSPTR is added to the fixed pointer stack in FUNAREA, and the displacement from the UCB to the pointer is stored in the parameter list. A scope (PARSCP) is resolved into two data structure pointers (DSPTR) by SCANNER (PUTINHIT). The DSPTRs are added to the fixed pointer stack in FUNAREA and the displacement to the

second pointer is stored in the parameter list. Unmatched optional parameters and dummy parameters are added to the parameter list (PAROPT). Dummy scopes and hits are filled with special DSPTRs, dummy window numbers are zeroed, implied insert points are set to X'0001' for ATTEND to fill in later, and regular unspecified optional parameters are left as zero. After all the parameters are checked (PARFINI), an incomplete function can be filled (FILLINC), or an unfinished insert functions, i.e. Insert and Substitute, can be completed (INSRT).

The function element is complete so CLINTERP branches back to get another function (START).

Error messages are handled in a separate routine (ERRMESS).

Note that the parameter type of a LOCATE is a literal not a scope. MAINLINE will call SCANMAIN and then SCANMAIN will call SCANNER to resolve the literal parameter into a DSPTR.

4.5 SCANNER

4.5.1

Function - SCANNER creates data structure pointers (DSPTRs) from text patterns and buffer displacements.

Entry Point - SCANNER

Return Codes - R15 contains the return code.
4 means that the pattern was not found
8 means the pattern was illegal
12 means unlightpennable text was matched
X'010000004' means pattern not found; the count was exceeded

What it changes - The display buffer and the WCB

Who it calls - DISPLAY, MAPCORE, CORMAP, SNAPSK, UNSNAPSK

Who calls it - CLINTERP and SCANMAIN and UNIFORM SUBSTITUTE

Error messages - none

Storage used - X'259' bytes via IEAFARTS. Extra storage for RING elements that describe display buffers are getmained and then freed before returning.

Registers used -
R15 WR work register
R13 SCANNER data area
R12 base register
R11 and R1 pointers to the parameter list
R7 AUCB pointer to the UCB
R10 RWCB pointer to the WCB
R9 PATTLEN length of the pattern
R8 PATT pointer to the pattern
R6 BUFFCH character in the buffer
R5 BUFF pointer in the buffer being scanned

Important flags -
PLAYUNIT in UNIT in the UCB. DEFLAGS in the WCB.

```
CNTL - scan control byte
ELLIPSNG EQU   X'80' set if doing second pattern in
an ellipsis
ONEIAND  EQU   X'40' set if one initial hit point in
pattern
TWOIAND  EQU   X'20' set if two initial hit points in
pattern
BLANPREV EQU   X'10' set if previous buffer character
was a blank
BACK     EQU   X'08' set if scanning backwards
SNAPFIRS EQU   X'04' set if first buffer is unsnapped
MIXEDMOD EQU   X'02' set if scan is upper and lower
case
SRCHING  EQU   X'01' set if trting for first
character
                     off if matching rest of the
pattern

CNTL1 - scan control byte 1
TWOTODO  EQU   X'80' two separate patterns to do
SECOND1  EQU   X'40' doing second of two patterns
NODISRST EQU   X'20' do not reset display before
second pattern
QUESORDR EQU   X'10' unsure of order of lphits

RBIT - flag in ring element
```

```
RSUCCESS EQU   X'80' successfull match in this buffer
RFIRST   EQU   X'02' first item in ring
RLAST    EQU   X'01' last item in ring
```

4.5.2 Parameters

On entry, R1 points to a 5 word block of pointers. The high order bits of the pointers may be used as flags.

PARM1 is either zero or points to a 10 byte data structure pointer (DSPTR). If not zero, DISPLAY is called with this data structure pointer, DSTART, so the search is started at this point (except as modified by the "place to start" information in PARM3).

PARM2 points to the pattern(s) to be used by SCANNER. The pattern is a two byte length (not necessarily halfword aligned) followed by the character string. If the length is zero, the DSPTR will point to the first character determined by PARM1 and PARM3. If the high order bit is zero only one pattern is to be used. If the high order bit is one, two independent patterns are expected (used at one time when scopes were specified as two patterns.) Both patterns are stored according to the above format; the second immediately following the first. The search for the second pattern begins at the first character following the hit point of the first pattern, unless overridden by PARM3. The pattern is never modified by SCANNER.

PARM3 points to a halfword aligned area containing information on where to actually start scanning relative to the start of the current buffer (if PARM1 is zero) or relative to the start of the buffer determined by PARM1 (if PARM1 is not zero).

The area pointed to by PARM3 is one or two halfwords long depending on the high order bit of PARM2. Thus there is one halfword of "where to start" information for each pattern given by PARM2.

The manner in which each halfword is interpreted depends on a bit in PARM3, the high order (X'80') bit for the first halfword and the next bit (X'40') for the second halfword. If the bit is zero the halfword

field is treated as a line number and a character number, each 8 bits (a sign bit and seven magnitude bits). The line number tells how many lines and in what direction to scroll before starting to scan. The character count tells how many characters to move the starting point in that line. If the character count is negative, its magnitude should be smaller than the current line length.

If the proper bit (X'80' or X'40') is set, the halfword is treated as a displacement from the beginning of the text in the buffer to the character. (This is for intelligent terminal data.) This option should not be used with a non-zero PARM1.

If two patterns are requested (via the high order bit of PARM2) and if the second halfword field of PARM3 is zero, the search for the second pattern starts at the character after the "hit point" of the first pattern. If the second field is not zero (e.g. non zero line number, character number pair or intelligent terminal data) then the fact that SCANNER is processing the second of two patterns is ignored and a hit point for the second pattern will be determined independently of the hit point of the first. (Thus a pair of hit points for a scope can be in the wrong order. The proper field is set in the returned DSPTR to indicate this.)

PARM4 points to one or two consecutive 12 byte area(s) in which the DSPTR(s) are returned. If the high order bit of PARM4 is not set, SCANNER will return one DSPTR for each pattern specified in PARM2. If the high order bit of PARM4 is set, SCANNER will return two DSPTRs for the pattern specified in PARM2. (If two patterns are specified each returning two pointers, an error is recognized. This is done because there is currently no reason to allow it.) For a discussion of what the DSPTRs point to, see 'What DSPTRs are returned'.

Actually DSPTRs are only 10 bytes long; the extra halfword is provided for possible inclusion of a relative pointer indicator.

If any bit in the high order byte of PARM1 is set, other than the highest, SCANNER knows it is not being called from CLINTERP. In this case if a normal forward scan is requested, the first character of the buffer is skipped. (This permits a repeat of a

pattern scan, such as Locate, to find successive occurrences.)

PARM5 was added in September 1973 for the multiple window version of FRESS. PARM5 points to one or two halfwords that are the window numbers associated with intelligent terminal buffer displacements given in PARM3. This parameter is used only when CLINTERP calls SCANNER with intelligent terminal data.

4.5.3 Valid Patterns

Patterns can have headers delimited by DLSPOT characters (DLSPOT is defined in FUNAREA as &) unless the pattern is zero, one or two characters long. Such short patterns cannot have headers. The presence of a header is denoted by an initial non literal DLSPOT character and the last character of a header is the next occurrence of DLSPOT. There will be an error if no second DLSPOT occurs. The part of the pattern excluding the header will be called the body of the pattern. It can have a length of zero. These headers are set up by whoever calls SCANNER or they may be included in a pattern by the user.

The header may include certain characters having defined meaning. Other characters are illegal in the header. An option and its opposite (e.g. mixed vs upper case) may be selected any number of times in one header. The last selection will be effective.

Legal characters in headers (either lower or upper case):

M Search is to be in Mixed mode. If a letter is lower case, the body of the pattern will match only the same lower case letter in the buffer. This is default for lp hits.

U Search is to be in Upper case mode. The case of the letter is insignificant, both in the body of the pattern and in the buffer.

This is default for locate operations.

B Search is to proceed backwards. The first character of the body must match a character before the start position of the scan as determined by PARMS 1, 2, and 3. The default is forward scan.

N Currently ignored. It used to be the delimiter at the beginning of a number.

0-9999 The number means that the search for the pattern is to proceed through nnnn hundred characters before failing.

Body of the Pattern - Each character in the body of the pattern (except DLSPOT and ellipsis) matches at least one character in the buffer. Each digit character in the pattern matches itself in the buffer. Depending on Mixed or Upper case modes, each letter will match either itself or the same letter in the opposite case. Each special character other than blank, DLSPOT, and the ellipsis notation will match only one occurrence of itself in the buffer. Blank will match one or more occurrences of itself in the buffer. If no blank occurs at the end of one line in the buffer, one is simulated,i.e. if the pattern has a blank at the right position, scanning will proceed to the next line in the buffer and the next character of the pattern; if the pattern lacks a blank, the current possible match fails and a different match is sought.

Blanks in the buffer at the beginning of lines are ignored completely (as in indented or centered text, at paragraphs, etc.)

ELLIPSIS - Three periods in the body of a pattern indicate ellipsis. (One or two periods are taken literally; four or more periods are taken as a literal occurrance of one fewer periods.) Ellipsis will match any sequence of characters. Currently the limit on the number of characters which ... will match is the search limit (i.e. the value which can be specified in the header) reduced by the number of characters which were searched over in locating the part of the body preceeding the ellipsis.

Only one ellipsis may be specified in the body of the pattern. It is not possible to indicate a literal period either directly before or directly after an ellipsis. The four periods will be taken together to mean three literal periods.

DLSPOT - DLSPOT in the body of the pattern is used to indicate what characters should be pointed to by the DSPTRs returned. They do not match characters in the buffer themselves; they merely single out characters in the pattern which are being pointed at. The DLSPOT should follow the character to be pointed at. Two DLSPOTs can occur in a row in a scope. (This could also be done with one DLSPOT if the character qualifier were applied to the function.) To request each single literal occurrence of DLSPOT in the body of the pattern three of them are required.

Non literal DLSPOT characters may not be the first character of the body of the pattern nor may they occur immediately to the right of an ellipsis.

4.5.4 What DSPTRs are returned

When called for two hits in one pattern (i.e. for a scope) the default is that the DSPTRs point to the beginning and the ending of the pattern. When two non-literal DLSPOT characters are present the body of the pattern, the buffer character matching the literal pattern character immediately to the left of each of the DLSPOT characters is indicated by the DSPTRs returned. If the two DLSPOTs are adjacent in the body, the character to their left is the one indicated (by both). When only one non literal DLSPOT character is present in the body, the two hits are the first character of the body, and the character indicated by the DLSPOT character.

When SCANNER is called from CLINTERP (for lp hits) different defaults with respect to returned lp hits are in effect than when called for a pattern scan. For lp hits - when only one hit is requested, the default is the last character of the body of the pattern. When a single nonliteral DLSPOT occurs, it indicates the character in the normal manner. (More then one DLSPOT is recognized as being illegal.) For the pattern scan - when only one hit is requested, the default is the first character the pattern matches. When DLSPOT is present, same as above. Note

that for a forward scan, the first character of the buffer is never examined.

In general when a blank is specified in the body of a pattern it will match more than one blank in the buffer. If n blanks occur in the pattern, they will match at least n blanks in the buffer, the first n-1 each matching one blank and the last matching at least one. If a DSPTR is returned for a blank in the pattern, the pointer indicates the first blank in the data structure which the pattern blank matched.

4.5.5 SCANNER Algorithm

Note that names in parentheses are labels in the program where the indicated action takes place or the name of a storage location.

Since SCANNER is too large to be addressed with one base register, part of it is a subroutine (SCA2SECT) with separate addressing. After initialization, the subroutine (SCA2SECT) is called. The subroutine checks for header information (INITP1), divides ellipsis (INITEL1) into two patterns, finds the appropriate WCB, and checks PARM3 for scroll information.

Then the buffer to scan is set up using PARM1 and PARM3. DISPLAY is called if there is a "place to start" DSPTR in PARM1 or if the number of lines to scroll is not within the current buffer (INIT4).

If the length of the pattern is zero (INITDONN), the pattern matching section of SCANNER is skipped. During a forward scan, DISPLAY is called to fill the buffer with the next block of text; then the buffer is searched for the first character of the pattern. During a backward scan, DISPLAY is called to fill the buffer with the previous block of text by specifying a scroll back; then the buffer is searched for the first character as in a forward scan. For each display buffer a RING element is created (GRING) to describe the buffer and the current state of the pattern match. The RING elements are linked together in a doubly linked list.

The buffer is searched for the pattern by doing a TRanslate and Test for the first character of the

pattern in the buffer (STARTSRC). If the first character is found, the displacement to it is saved. Then SCANNER tries to match the rest of the pattern (MATCHING).

If the end of the buffer is reached while searching for the first character (SRCEOB), the number of characters to search is checked (FAILCNT). A new buffer is filled by DISPLAY (NEXTBUF or PREVBUF) unless the beginning or the end of the text is reached (FAILURE). Note that for a backward scan, if the search has not reached the last RING element, then the links on the RING elements are reversed, i.e. a forward link points to the RING element associated with the previous buffer (SRCEOB3).

If the end of the buffer is reached while matching (MEOBUFF), the matching continues in the next buffer.

If matching fails (MAT51), the RING elements and the pattern pointers are reset and the search resumes for the first character of the pattern.

If the pattern matches (EOPATT) for a forward scan, then the search is complete (SUCCESS). For a backward scan, if a match is found, RSUCCESS is set in RBIT in the RING element and the displacement to the pattern is saved. The scan continues until the end of the buffer is reached, then the RING element is checked to see if a successful match occurred. If so the search is complete (SUCCESS).

Next SCANNER resets the "place to start" DSPTR and branches back to finish an incomplete ellipsis (INITDONN).

Then the DSPTR(s) are stored in the space pointed to by PARM4, any left over ring elements are freed (RETNA2), and if necessary, DISPLAY is called to reset the display to its original position. If there are two independent patterns, then SCANNER repeats for the second one (STARTSND).

5 DISPLAY

5.1 OVERVIEW

The section of FRESS collectively known as Display takes care of scrolling and generating a buffer and correlation map from the data structure. The buffer contains the next n lines of the online display in a particular window, n being determined by the type of terminal being used. The correlation map correlates a place in the buffer with a position in the data structure so that subsequent pattern scans in the buffer can be translated into DSPTR's.

The following diagram illustrates the flow of control among the display programs.

The following list indicates the general function of each Display routine.

DISPLAYs-
ets up
pointers
to
control
blocks,
decides
what is
being
displayed
(text,
label
space,
keyword
space),
calls

appropriate
routine

MAKEBUFF-
moves
regular
text into
the
buffer;
calls
ORDERS to
handle

structure,
ONLINE to
handle
format
code
ORDERSde-
termines
what type
of
structure
display
is at and
if any
action
need be
taken
(e.g. on
jumps);
calls
other
routines
to
actually
move the
structure
into the
buffer
ONLINEmo-
ves
format
codes
into the
buffer,

interpre-
ting them
if

necessar

SCOPFIND-
calls
BLOCSCAN
and
DECLBSCN
to fill
in SCOPE
control
block;
called
after

non-linear
travels
to get

information
about
where
display
is

DECLBSCN does decimal label level calculat- ion by scanning the structure space; can retrieve a block given its declab number or find the declab number of a block given its internal label; follows splices and keyworded jumps within a single file while counting

ENDLINEe-
nds the
current
buffer
line by
putting
in any
needed
carriage
control;

determines
if
scrolling
done or
buffer
filled;
if so,
returns
to
whoever
called
MAKEBUFF
(either
DISPLAY
or

SCRLBACK).
See below
for
further

information
NEWLINEb-
egins a
new line
in the
buffer by
adding
any
necessary
carriage
control

JUSTIFYp-
ads the
current
display
line with
blanks to

right-ju-
stify i
NEWLABma-
kes
non-line-
ar
travels
for
ORDER
GETENTRY-
gets an
entry in
splice
return
stack or
instance
return
stack
MAPROUTc-
onsists
of entry
points
MAKEMAP1
and
MAKEMAP2
which
make
first and
second
level

correlation
map
entries
(see
below)

MOVEORDR-
moves
structure
orders
(percent
signs,
opcodes,
and
modifier-
s) into
the
buffe
VIEWMOVE-
moves
viewspecs
on jumps
into the
buffer
MOVELABm-
oves
labels on
points
and
blocks
into the
buffer
MOVEKEYm-
oves
keywords
on blocks
and tags
into the
buffer
SCRLBACK-
handles
scrolling
backward-
s; see
below for
more
detailed

descript-
i

KEYBUFFh-
andles
display
of and
scrolling
in the
keyword
spac
LABLBUFF-
handles
display
of and
scrolling
in the
label
space

DISPLAY is called to do a task on a single window. All Display routines are independent of whether a single or multiple window version is being run. They see only one window at a time in either case.

DISPLAY determines the task being requested by the status of certain locations in the WCB (window control block). If DSCRLL is non-zero, a scroll is being requested. If the high-order bit of DSCRLL is on, the other 7 bits represent the number of lines to scroll backwards. If the bit is not on, the scroll is to be done forwards. DFLAGS indicates additional information about where in the data structure the buffer is to be started. See Section 5.6.1 for an explanation of each of these flags.

5.2 FILLING THE BUFFER AND SCROLLING

If the main text, structure, or annotation space is being displayed, DISPLAY then determines where in the data structure the display is to begin. If DFLAGS10 is on, it means display is to begin at the structure corresponding to the internal label in DINTLB. This occurs after any non-linear travel function is used. If DFLAGS9 is on, display is to begin at the bottom of the file, defined to be one display line before the end area order. If DFLAGS1 is on, display is of the picture whose number is in DINTLB. If none of these flags is on, display starts at the file, page, and displacement indicated in DSTART. If DFLAGS4 is not on, DISPLAY must call SCOPFIND to find out some information about the

'scope' (see Section 5.6.2) of the first character to appear in the buffer.

DISPLAY then calls MAKEBUFF to fill the buffer. MAKEBUFF scans through the data structure, moving text into the buffer. When enough text has been moved to fill one line, the length of which is determined from DLENGTH, ENDLINE is called to end that display line by adding any carriage control or padding necessary (determined by the characteristics of the terminal). Then NEWLINE is called to add any necessary carriage control to the start of the next line. The lines in the buffer are of variable length in most cases; that is, the carriage return goes immediately after the last word which can be fit on that line, and is followed immediately by the next buffer line.

When MAKEBUFF encounters a format code or structure order, it calls ONLINE or ORDERS respectively to handle them. These routines start new lines if necessary by calling ENDLINE and NEWLINE, and move the codes into the buffer. They have various auxiliary routines which they call as well.

Scrolling works the same way except that the first line of the buffer is written over and over again instead of filling the buffer. To do this, ENDLINE changes the pointer to the current buffer position to point to the start of the buffer instead of to the position where the next line should go.

There is one major irregularity in Display's flow of control. When the buffer is filled, or scrolling forward or "count scrolling" (see Section 5.5) is completed, this is discovered by ENDLINE. At this point, rather than returning through the calling routines indicating the task is completed, ENDLINE returns directly to the routine which called MAKEBUFF (either DISPLAY or SCRLBACK). There is an entry point in ENDLINE called CHEKDONE which is used by a few other routines (MAKEBUFF, NEWLAB) to see if the task is done. There is a standard sequence used by these routines for abruptly ending the buffer if an error is found. The code for this is in a COPY called ALLDONE.

Since ENDLINE may thus "go away" at any time, all routines must be careful to have all flags, etc., set correctly when passing control to another routine. For example, if while a scroll is being done, MOVEKEY finds it necessary to split a keyword string over 2 lines, it must set a flag indicating it is in the middle of

displaying a keyword data field when it calls ENDLINE. Then if ENDLINE returns to DISPLAY because the scroll is completed, the flag will be on to show that the new buffer being created from that point starts in the middle of a keyword field. If instead ENDLINE returns to MOVEKEY, the flag must then be turned off.

Besides all the common control blocks which are used to transfer information, certain registers are always set to point to particular things. These registers are set in MAKEBUFF and are used by MAKEBUFF, ONLINE, ORDERS, MOVEORDR, VIEWMOVE, MOVEKEY, MOVELAB, NEWLAB, GETENTRY, ENDLINE, and NEWLINE. The register assignments are:

```
     R4 -> current position in data structure
           (often EQU'd to VARPTR)
     R5 -> WCB
     R7 -> UCB (as everywhere in FRESS)
     R9 -> current buffer position
    R11  =  number of characters  already put on current
           display line.
```

5.3 CORRELATION MAP

When a buffer is generated, Display also generates a correlation map which correlates certain positions (displacements) in the buffer to places in the data structure. When SCANNER matches a context pattern from a command line with a string in the buffer, a DSPTR can be formed (by CORMAP) to point to the corresponding place in the data structure. There is a separate correlation map for each window. Each map is found by using the contents of DCMAPLOC in the WCB as a displacement from the top of the UCB to the map.

The correlation map has two kinds of entries: first level and second level. The first level entries, represented by control block CORRMAP1, are made for the first character in the buffer, every time a non-linear jump is taken (e.g. after a splice), and every time a new page is encountered. They indicate major discontinuities in the data structure being represented in the buffer. The form of one of these entries is:

```
CORRMAP1 DSECT
MAPBUFF1 DS    H     disp  from  top  of  UCB  to buffer
                                 location being correlated
MAPDISP1 DS    H     disp to next first level entry
```

```
MAPWIND  DS    X     window number
MAPPROT  DS    X     protect function byte
MAPFILE  DS    H     file number
MAPPAGE  DS    H     page name
         DS    H     unused (for alignment)
```

Second level entries, represented by control block CORRMAP2, are made at any discontinuity in the data structure or buffer. Entries are thus made at all places where first level entries are made, plus at the beginning of every display line, at the end of every display line if carriage control (e.g., a carriage return) is necessary, when a structure order is displayed, and after a structure order is displayed. The form of a second level entry is:

```
CORRMAP2 DSECT
MAPBUFF2 DS    H     disp from top of UCB to buffer
                           location being correlated
DISPJUMP EQU   x'80' display   jump taken (includes
                           splices, instance returns,
                           and block trail jumps)
MAPORDER DS    H     disp in page to character being
                           correlated
GENTEXT  EQU   x'80' generated text (used for decimal
                           labels,    area    lines,
                           picture names). See below
                           for meaning.
```

The block of second level entries referring to a particular page in the data structure follows immediately after the first level entry for that page.

Correlations are fairly straightforward except when displaying structure orders. In this case, correlation depends on the desires of the Editing routines. The percent sign and opcode representation (i.e., L for Location, J for Jump, S for Splice, etc.) are correlated to the x'6C' of the order in the data structure. The representation of a "first modifier" (annotation symbol is the only one implemented; the category was meant to include table of contents and index symbols) is also correlated to the x'6C'. A 'second modifier' (the second letter in the representation of splice and ecilps orders, P and C respectively) are correlated to the modifier byte of the order in the data structure. This is to allow a splice or ecilps to be turned into a jump or pmuj by "deleting" the modifier which makes it a special case. For the same reason, the decimal number of a decimal label block is correlated to the modifier

byte. This allows the user to delete the special characteristic and turn the decimal block into a "normal" block by deleting the decimal number. However, the decimal number of a decimal label reference tag is correlated to the x'6C' of the order since there is no such thing as a "normal" tag; deleting the decimal number can only mean to delete the whole tag.

Correlating area lines is another special case. The character string placed in the buffer to represent an area line is known as "generated text". Therefore the whole string correlates to the x'6C' of the order.

To find a DSPTR from a buffer displacement, CORMAP first finds two consecutive first level entries such that the buffer disp falls between them, or it finds that the buffer disp is after the last first level entry. Then it searches the second level entries between these two first level entries until it finds two consecutive entries such that the buffer disp falls between them. (Again it may find the buffer disp is after the last second level entry in that section.) Using the first of these second level entries, CORMAP now computes the exact displacement. If the GENTEXT flag is on, MAPORDER is the correct displacement. If the flag is not on, MAPBUFF2 is subtracted from the buffer disp being correlated, and the difference is added to MAPORDER to get the correct displacement down the page to the character.

Correlation map entries are made by MAKEMAP1 and MAKEMAP2, entry points in MAPROUT. These routines are called by any routines which place text in the buffer - MAKEBUFF, ONLINE, NEWLAB, ORDERS, MOVEKEY, MOVELAB, VIEWMOVE, MOVEORDR, ENDLINE. MAKEMAP1 is called by only the first four of these; MAKEMAP2 is called by all of them.

The information which these two routines need to make the entries is stored in DATD, the common control block for most of Display. The locations used are:

```
BUFFMAP  -> buffer location
DATAMAP  -> data structure location
PAGENAME number of page being handled
FILENO   number of file being handled
PROTBYTE current protection byte
PAGEADDR -> top of page being handled
```

The correlation maps are also used by MAINLINE to determine if a window should be regenerated after an

edit. The page numbers in the first level entries are compared against the altered pages (MODIFY flag on). If any match, the window is regenerated.

5.4 STACKS AND THE STACK CONTROL BLOCK

There are five important stacks in FRESS - the Jump Return Stack (JRS), Block Trail List (BTL), Memory Return Ring (MRR), Splice Return Stack (SRS), and Instance Return Stack (IRS). The first three are regulated by non-Display routines, although the BTL is used by Display. They are anchored in the UCB. The other two are used and maintained solely by Display routines.

However, the storage management for all five stacks is the same. They are allocated in the Stack Control Block (SCB) which is part of the UCB. Its location is given in SCBDISP in the UCB as a displacement from the top of the UCB. Each stack entry is 18 bytes long. (Only the JRS and IRS entries need 18 bytes, but for uniformity they are all the same length.) Entries are referred to by a number indicating how far into the SCB they are. The first entry is immediately after the 4 bytes of control information at the top of the SCB and the others follow sequentially. To find the displacement from the top of the SCB to entry n, the formula is

(n*18)-14

The four bytes of control information at the top of the SCB (explained further below) are:

```
SCBNEXT  DS    X     index of next available element
SCBMAX   DS    X     index of max allowaable element
SCBFREE  DS    X     index to free element list
         DS    X     unused
```

SCBNEXT is initially set to 1. It is incremented by 1 each time a previously unused element is allocated. When an element is no longer needed, it is added to the top of the free list pointed at by SCBFREE. The last byte of each 18 byte entry in the free list points to the next free element. When a routine needs another entry, it checks this free list first and uses the top element if one exists; if not, it uses the element pointed at by SCBNEXT. SCBMAX is the maximum allowable

element given how large the SCB is. This is set by MAINLINE when the SCB is allocated and is used for comparison each time SCBNEXT is used to get a new entry. If SCBNEXT is greater than SCBMAX, the message STACK CONTROL BLOCK ALL FILLED is printed. This can happen when any one of these five stacks requests a new entry. Stack elements can be popped by eliminating a block trail or memory return ring elements, or by doing a series of RETURNs to pop JRS entries. The number of SRS and IRS entries is controlled by what is being displayed, and thus can be changed by displaying from a different point.

5.4.1 Splice Return Stack (SRS)

The splice return stack is used by Display to keep track of which splices and keyworded jumps have been followed in scrolling through a file or files. Then if the user scrolls backwards and encounters the pmuj or ecilps at the other end, Display will retrace his path. The entries are made in ORDERS after the jump or splice is taken, but only if Display is performing a scroll, not a "count scroll" or generating the buffer. The next available element is gotten by a call to GETENTRY.

The form of an SRS entry is:

```
SRS      DSECT
SRSVIEW  DS    CL7   viewspec string in effect before
                                    splice
         DS    X     unused
SRSRFIL  DS    H     file no. where jump is
SRSRLAB  DS    H     intlab of jump
SRSTFIL  DS    H     file no. where pmuj is
SRSTLAB  DS    H     intlab of pmuj
SRSIBKW  DS    X     index of previous entry
SRSIFWD  DS    X     index of next entry
```

If a pmuj is encountered while scrolling backwards, its internal label is compared with the SRSTLAB of the top entry of the stack (pointed to by DSRS in the WCB). If they are the same and the file numbers also match, an "unsplice" is done; that is, scrolling backwards is continued from the place indicated by SRSRLAB and SRSRFIL.

The SRS is deleted and started over whenever a non-linear travel is done by the user (as opposed to a non-linear travel done by Display while scrolling or filling the buffer). Thus a Get Label, the creation of a block trail, or following a jump will create a new SRS.

5.4.2 Instance Return Stack (IRS)

The instance return stack is used by Display to keep track of which annotation tags have been followed to their respective annotation blocks. When Display encounters the end of the annotation block reached in this manner, it returns to the tag referencing the block. The entries are made in ORDERS after the annotation block has been reached. Again, the element is obtained by calling GETENTRY. An element is popped from the stack when the corresponding block end is reached.

The form of an IRS entry is:

```
IRS      DSECT
IRSVIEW  DS    CL7   viewspecs in effect at tag
IRSISRS  DS    X     index of associated SRS element
IRSRFIL  DS    H     file no. where tag is
IRSRLAB  DS    H     intlab of tag
IRSTFIL  DS    H     file no. where annot block is
IRSTLAB  DS    H     intlab of block
IRSIBKW  DS    X     index of previous entry
IRSIFWD  DS    X     index of next entry
```

Some of the fields of an IRS entry are redundant. For example, IRSTFIL and IRSRFIL are always the same since annotation references are always within a file; IRSRLAB-2=IRSTLAB by definition of annotation tags. The reason for this redundancy is that the IRS was originally designed to handle splices to blocks as instances; all the information was necessary since splices to blocks could be interfile and there was no standard relationship between the intlab of the point which anchored the splice and that of the block which anchored the ecilps.

As indicated above, an IRS entry contains a reference to the "associated" SRS element. This means the index of the top of the SRS when the IRS entry was made. When the IRS entry is popped from

the stack, any SRS entries made since the IRS entry was made are popped as well. The need for this is not entirely clear. It means that when scrolling backwards if any of the ecilpses within the annotation block are encountered, they will not be followed to the splices at the other end. However, if in scrolling backwards, the tag is not followed to the end of the referenced block, the top of the SRS will in fact refer to the first ecilps to be encountered. This can happen, since tags are followed when scrolling backwards only if the associated keyword string is satisfied. It's a rather ambiguous situation because following ecilpses while going backwards depends only on the path taken when scrolling forward, while following annotation tags while going backwards depends only on the associated keyword string.

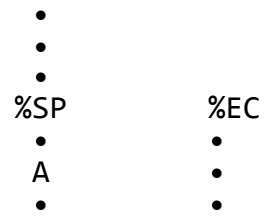
IRS entries, unlike SRS entries, are made in all cases, whether scrolling, count scrolling, or filling the buffer. However, only when scrolling is the real IRS used. That is the only case when the status of the stack is important after the task is completed. If DISPLAY is called to fill the buffer, the current IRS is copied into a temporary area (the TIRSCB) and all new entries are made in this temporary stack. If a scroll back is being done, SCRLBACK copies the current IRS into a temporary stack before it calls MAKEBUFF to count scroll.

5.5 SCROLLING BACKWARDS

When DISPLAY determines that a backwards scroll is desired, it calls SCRLBACK. SCRLBACK then scans backwards in the data structure one character at a time from the position indicated in DSTART until it finds something which starts a new line (a jump, block start, area line, or format code). It then transfers the information in DSTART to DFINIS and sets DSTART to point to the character it backed up to. SCRLBACK then calls MAKEBUFF to count the number of display lines between DSTART and DFINIS. This is called "count scrolling".

Count scrolling is handled the same as scrolling in most cases. The only differences occur when deciding whether or not to take a jump or splice while counting. If the user has scrolled back over a splice (but not by scrolling over the ecilps), the splice should not be

taken when count scrolling forward. For example, given the following structure:



If the user arrives at point A by some non-linear means, e.g., a Get Label, and scrolls backwards over the splice, ORDERS should not take the splice when count scrolling or it will never get back to point A.

ENDLINE determines when the count scrolling is completed by comparing DFINIS to the pointer to the current position in the data structure when it is called to end a display line.

When control returns to SCRLBACK, it determines if enough lines have been scrolled backwards. If not, it continues scanning backwards in the data structure in the manner indicated above. If at least enough lines have been scrolled back, SCRLBACK returns to DISPLAY. If just enough lines have been scrolled back, DISPLAY then calls MAKEBUFF to fill the buffer from the position in DSTART. If too many lines have been passed, DISPLAY first calls MAKEBUFF to scroll forward the difference, then calls MAKEBUFF again to fill the buffer.

There are many irregularities which have to be considered when implementing this algorithm. The code to handle these irregularities is scattered throughout the Display routines and this causes some difficulty when trying to modify code.

5.6 COMMON CONTROL BLOCKS

There are a number of control blocks which Display programs use extensively. All except one of these, the last discussed, are common with the rest of FRESS as well. The global use of these control blocks is explained, as well as the specific use of each field.

5.6.1 Window Control Block (WCB)

The WCB contains all the necessary information about a window (or buffer) and the text it contains. This includes the dimensions of the buffer (line length, number of lines), displacement to the correlation map and buffer, flags and data structure pointer telling DISPLAY where to start generating the buffer, "scope" of the first character in the buffer (see Section 5.6.2), and some fields used internally by Display routines. Following is a list of each of these individual fields.

```
WCB      DSECT
DWINDN   DS    C     window number
DLINES   DS    C     max no. of lines in this window
DLENGTH  DS    H     max length of each line
DCWIND   DS    C     something to do with coupling
                           windows,         not
                           implemented
DFWIND   DS    C     something to do with coupling
                           windows,         not
                           implemented
DSCRLL   DS    C     no. of lines to scroll if
                           scrolling
SCRLBIT1 EQU   x'80' on if scrolling back
DFLAGS   DS    2C
* first byte of DFLAGS:
DFLAGS1  EQU   x'80' displaying from picture; pict
                           no. in DINTLB
DFLAGS2  EQU   x'40' do not display any text
DFLAGS3  EQU   x'20' display msg (checked in low
                           level attn for
                           sending)
DFLAGS4  EQU   x'10' regen old display or scrolling
                           (i.e., scope already
                           set up)
DFLAGS5  EQU   x'08' set in SCANNER for unknown
                           reason
DFLAGS6  EQU   x'04' scrolling
DFLAGS7  EQU   x'02' count scrolling
DFLAGS8  EQU   x'01' display space from bottom
* second byte of DFLAGS:
DFLAGS9  EQU   x'80' use space number for starting pt
DFLAGS10 EQU   x'40' use internal label in DINTLB for
                           starting pt
DFLAGS11 EQU   x'20' used in SCRLBACK to indicate hit
                           decimal block so
                           have to call
```

```
                                   SCOPFIND   to  count
                                   declevels again
DFLAGS12 EQU   x'10' doing a Type
DFLAGS13 EQU   x'08' count    double   spacing   when
                                   scrolling
DFLAGS14 EQU   x'04' tells   ATTNxxxx  to  ship  this
                                   window;    set    by
                                   MAINLINE    usually.
                                   Also  used  to  tell
                                   ORDERS not to take a
                                   particular jump when
                                   count scrolling
DFLAGS15 EQU   x'02' unlightpennable  text  in window
                                   (i.e.,    Query   or
                                   Help)
DFLAGS16 EQU   x'01' unknown
DBLANK   DS    C     ???
ORDDISP  DS    H     set  by  ENDLINE  to  disp  down
                                   current    page   to
                                   VARPTR   (r4)   when
                                   finished   scrolling
                                   or typing; if buffer
                                   ended  in  middle of
                                   an   order,   VARPTR
                                   will  still be pting
                                   at  the  percent  of
                                   the  order; also set
                                   by   SCOPFIND   from
                                   ORDERPTR in SCOPE
DIRS     DS    C     index to top (most recent entry)
                                   of IRS
DSRS     DS    C     index to top (most recent entry)
                                   of SRS
         DS    C     was DBTL, moved to UCB
         DS    C     was DJRS, moved to UCB
         DS    C     was DMRR, moved to UCB
DSPACE   DS    C     space number
DINTLB   DS    H     if  DFLAGS1 set,  is picture no.
                                   If  DFLAGS10 set, is
                                   intlab   to  display
                                   from
DSTART   DS    14C   posn  for  display  to  start if
                                   nothing      special
                                   happening
         ORG   DSTART
DBFILE   DS    H     file number
DBVIEW   DS    7C    viewspec string in effect
DBFLAGS  DS    C
DBFLAG1  EQU   x'80' endline at start of buffer
DBFLAG2  EQU   x'40' startline at start of buffer
DBFLAG3  EQU   x'20' unused
```

```
DBFLAG4  EQU   x'10' transcribe doing a return
DBPAGE   DS    H    page number
DBDISP   DS    H    disp down page
DFINIS   DS    14C  used for communication between
                          SCRLBACK and
                          ENDLINE. SCRLBACK
                          goes backwards at
                          DSTART, sets DSTART
                          to where stopped
                          going back, sets
                          DFINIS to original
                          DSTART. ENDLINE
                          knows to stop when
                          have gone forward to
                          DFINIS.
         ORG   DFINIS
DEFILE   DS    H    file number
DEVIEW   DS    7C   viewspec string in effect
DEFLAGS  DS    C
DEFLAG1  EQU   x'80' endline in buffer
DEPAGE   DS    H    page number
DEDISP   DS    H    disp down page
BUFFRLEN DS    H    real length of buffer and corr
                          map (constant)
DCLINES  DS    H    no. of lines actually in buffer
DMESSLOC DS    H    unused now
DMESSLEN DS    H    unused now
DBUFFLOC DS    H    disp to buffer
DBUFFLEN DS    H    length of same (after being
                          filled)
DCMAPLOC DS    H    disp to corr map
DCMAPLEN DS    H    length of same (after being
                          filled)
ATIRSCB  DS    F    -> temporary IRS control block
         DS    0D
DSCOPE   DS    94C  SCOPE block (see below)
```

5.6.2 Scope

The SCOPE Dsect contains information about a particular character in the data structure. The SCOPE macro has two forms. The first expands into twelve bytes of information which Editing routines use to determine the characteristics of a hit point - whether it is in a structure order, what area it is in, the internal label of the preceding structure order. This information is all filled in by BLOCSCAN.

The second form of the macro, gotten by specifying ROUT=DISPLAY, expands into 94 bytes, including the twelve bytes discussed above. This is the version of SCOPE filled in when SCOPFIND is called by various Display routines. SCOPFIND calls BLOCSCAN to fill in the first section, and DECLBSCN to fill in the fields indicating the decimal label level of the character for which the scope is being done. SCOPFIND is called whenever DISPLAY needs information about the first character in the buffer (e.g. after the user has done a non-linear travel) and when Display does a non-linear travel in the course of scrolling or filling the buffer. A dummy SCOPE is filled in when filling the buffer since the real SCOPE, which is part of the WCB, is filled in for the first character of the buffer.

There are a lot of unused fields in the Display section of SCOPE but these have been left in to preserve the positioning of the fields at the bottom. Following is a list of all the fields in SCOPE:

```
SCOPE    DSECT
BLCSCNBT DS    0F
SCANTYPE DS    X     flag indicating what type of
                           scan BLOCSCAN is
                           doing; see BLOCSCAN
                           listing for further
                           information
MSCOPES  EQU   x'80' editing routines
LSCOPES  EQU   x'40' normal display
SPLITP   EQU   x'20' split page BLOCSCAN
DLABSCAN EQU   x'10' Display only wants last dec lab
ALTRSCAN EQU   x'08' previous alter formatting scopes
TCONTROL EQU   x'04' unused currently
BEFORE   EQU   x'02' end BLOCSCAN for split page
                           before midpoint is
                           reached
AFTER    EQU   x'01' end BLOCSCAN for split page
                           after the midpoint
ORDERF   DS    X     flags to indicate if hit 5 bytes
                           of order
PERCENTF EQU   x'80' hit percent sign of order
OPCODEF  EQU   x'40' hit opcode of order
MODFIERF EQU   x'20' hit first modifer bit
INTLAB1  EQU   x'10' hit 2nds modifier bit
INTLAB2  EQU   x'08' hit 3rd modifier bit
ZEROF    EQU   x'04' hit 00 of a data field
LZEROF   EQU   x'02' hit x'00' of last data field
JE6CF    EQU   x'01' unused; formerly meant hit %% of
                           jump menu
```

```
TEXTTYPE DS    X     type of non-regular text hit
JUMPTO   EQU   x'80' middle of jump-to explainer
JUMPFROM EQU   x'40' middle  of  jump-from  explainer
                                 (pmuj)
TOFCINDX EQU   x'20' unused now
SKECHLNK EQU   x'10' unused now
DELETED  EQU   x'08' unused now
FIELDS   DS    X     type of data field; flags set by
                                 Display  routines if
                                 end a line in middle
                                 of data field
LABELF   EQU   x'80' label field
KEYWORDF EQU   x'40' keyword field
VIEWSPCF EQU   x'20' viewspec field
SKETCHF  EQU   x'10' sketchpad name field
TEXT360F EQU   x'08' format code
IRSF     EQU   x'04' unused now
DLABREFF EQU   x'02' decimal label reference field
ANNOTF   EQU   x'01' annotation tag reference field
TYPETEXT EQU   TEXTTYused  under  another  name  some
                                 places
AREANUMS DS    H     area number of last area
LASTINTS DS    H     intlab of last structure
         DS    XL3   free
ORDOPCOD DS    X     for BLOCSCAN to play with
ESCOPLEN EQU   *-SCOPend   of  Editing's  section  of
                                 SCOPE
LENPROT  DS    H     unused
USCRSTRT DS    F     underscore starting address
PROTSTK  DS    CL40  unused
LNTXTKY  DS    H     unused
DTXTKY   DS    H     unused
LNJTXTKY DS    H     unused
DJTXTKY  DS    H     used   in   ORDERS   under  name
                                 SAVELEV to save decl
                                 level of dec lab ref
                                 tag
PTAB     DS    X     current   tab   number  (may  be
                                 unused)
DECLEVEL DS    X     decimal label level
DECLINES DS    X     no.  lines  displayed  for  last
                                 dlab  (unimplemented
                                 currently)
LINESKIP DS    X     number  of  lines  left to skip;
                                 set             when
                                 interpreting  format
                                 codes     in    case
                                 ENDLINE goes away in
                                 middle
         DS    0H    next   field  must  be  halfword
                                 aligned
```

```
DECLABEL DS    CL16  current   decimal  label;  8  HW
                                 fields    containing
                                 binary   number  for
                                 each dec level
HNGINDNT DS    X     current   hanging  indent  level
                                 (may be unused)
LMARGIN  DS    X     unused
CODESKIP DS    X     number of lines this format code
                                 skips   (used   with
                                 LINESKIP above)
LWIDTH   DS    X     unused
FMTFLAGS DS          flags   for   formatting;   some
                                 indicated   as   used
                                 may be unused
USCORE   EQU   x'80' underscore
MLTICAPF EQU   x'40' multicap flag
DBLSPACE EQU   x'20' double space
DKEEP    EQU   x'10' unused
PARAFLAG EQU   x'08' unused
ASISFLAG EQU   x'04' as is flag
SKIPFLAG EQU   x'02' skip flag
DCOLUMN  EQU   x'01' unused
FMTFLAG1 DS    X     additional byte of flags
PSRET    EQU   x'80' DISPLAY   called  after  pattern
                                 scan or return
DSCOPLEN EQU   *-SCOPend   of  Display's  section  of
                                 SCOPE
```

5.6.3 DATD

DATD is a piece of storage shared among all Display routines except LABLBUFF, SCRLBACK, SCOPFIND, DISPLAY, KEYBUFF, and DECLBSCN. It is used both as temporary storage and as a means for communicating between the various routines. Although SCRLBACK does not share the same DATD with all these other routines, it has its own temporary storage which it also calls DATD and which has may of the same names contained in it as the global DATD. Following is a list of all the fields in DATD and the use of each one. There are many unused fields here, but all Display routines would have to be reassembled in order to take these fields out.

```
DATD     DSECT
SAV001   DS    18F   MAKEBUFF's savearea
```

```
WCBPTR   DS    F     to temporarily hold -> WCB while
                           register   is   used
                           (MAKEBUFF)
LINEPTR  DS    H     passes no. of chars on line from
                           ENDLINE  to  JUSTIFY
                           who    changes    it
                           according to the no.
                           of blanks added
SCBPTR   DS    F     ->  stack control  block, set by
                           MAKEBUFF,   used  in
                           ORDERS
TEMPPTR  DS    F     in MAKEBUFF  and ORDERS for temp
                           storage
IRSPTR   DS    F     -> IRS entry working on, ORDERS
SRSPTR   DS    F     used  as parm  in ORDERS calling
                           GETENTRY
UNITPTR  DS    F     -> current PLAYUNIT
LINENO   DS    H     ctr  of  no.  of  lines actually
                           generated  in buffer
                           working on - updated
                           by ENDLINE
LINES    DS    1C    max  no.   of  lines  in  buffer
                           currently generating
                           (used  in ENDLINE to
                           see  when  done with
                           buffer)
PAGEADDR DS    F     holds  -> top  of page currently
                           using
SCOPEPTR DS    F     ->  current scope  block; set by
                           MAKEBUFF  from  parm
                           from DISPLAY
SCOPFLGS DS    4C    unused
FCBSAVE  DS    F     MAKEBUFF  saves  -> current FCB,
                           ENDLINE     restores
                           when   through  with
                           buffer    in    case
                           changed files
CRNTFLD  DS    F     unused
PROTECTN DS    40C   used  to  save user's protection
                           string.    Set    in
                           MAKEBUFF.   PROTBYTE
                           also     saved    in
                           corrmap  entries  in
                           MAPROUT.      Others
                           never accessed.
         ORG   PROTECTN
PASSWRD  DS    7C
PROTBYTE DS    1C
PROTMASK DS    32C
DISPLBIT EQU   x'80' if on (in PROTECTN+8) everything
                           is   invisible  text
```

```
                                        (equivalent       to
                                        leftmost    bit   in
                                        protection     mask,
                                        which    does    not
                                        correspond   to  any
                                        particular function.
                                        Unknown   who  might
                                        set   it   or   why,
                                        except  that  is off
                                        if    specify    the
                                        standard    password
                                        string NONE.)
*  next  group  of flags  is set  up in MAKEBUFF when
routine  *first called and  every time ORDERS returns
with  a  return code  *of 4  instead of 0, indicating
have done non-linear travel
SUPERSCN DS    1C    controls   what   displayed.  If
                                        anything   on,  most
                                        routines      return
                                        without        doing
                                        anything.
DISPROT  EQU   x'80' on if DISPLBIT is on
INVTXKEY EQU   x'40' unused
DLEV     EQU   x'20' on   if  current  decimal  label
                                        level too deep to be
                                        visible;    set   in
                                        MAKEBUFF on basis of
                                        DECLAB    field   in
                                        VIEWSPEC string (not
                                        implemented)
DLINE    EQU   x'10' on    if    already    displayed
                                        allowable   no.   of
                                        lines/dec level (not
                                        implemented)
TTYPE    EQU   x'08' on  if viewspec  string says not
                                        showing regular text
                                        and are not starting
                                        display in an order
SCANORDR EQU   x'04' tested  in   ORDERS  to  see  if
                                        skipping over ORDER,
                                        but   never   set  by
                                        anybody
SCANEXPL EQU   x'02' tested  in   ORDERS  to  see  if
                                        skipping  over order
                                        but   never   set  by
                                        anybody
MISCFLAG DS    2C    miscellaneous flags for Display
* first byte of flags
FIRSTIME EQU   x'80' unused
SPLBIT   EQU   x'40' set    in   ORDERS   when   have
                                        determined that jump
```

```
                                    is to be taken -
                                    means should not
                                    start new line for
                                    pmuj and should take
                                    splice after
                                    displaying explainer
FIRST    EQU   x'20' has multiple meanings - see
                                    description after
                                    rest of fields
CORRMAP  EQU   x'10' on if need correlation map
ENDMULT  EQU   x'08' used under name MULTICAP in
                                    MAKEBUFF to indicate
                                    text string being
                                    handled starts with
                                    at least 2 capitals
ENDAREA  EQU   x'04' in ORDERS, on if end of page is
                                    also end of space,
                                    no area order
                                    present
VIEWBIT  EQU   x'02' in ORDERS indicates have hit
                                    viewspec data field
                                    so will change
                                    viewspecs later
JUMPF    EQU   x'01' in ORDERS indicates whether jump
                                    or pmuj reached
* second byte of flags
MIDMULT  EQU   x'80' middle of a multicap field which
                                    splits over a page
                                    boundary, MAKEBUFF
SWTCHPAG EQU   x'40' MAKEBUFF; on if on 2nd page of
                                    multicap field split
                                    over page boundary
DBLNTRY  EQU   x'20' unused
*  bits  used  by  SCRLBACK  - these  are not in this
MISCFLAG,  but  *in  flag  by  same  name in SCRLBACK
storage
DECFLAG  EQU   x'80' unused
IRSPOP   EQU   x'40' popping IRS, not SRS
FIRST    EQU   x'20' area line are at is start area
SCLCHEAT EQU   x'10' has something to do with getting
                                    scrolled lines out
                                    of format codes
                                    which cause skips
MDFLAG   EQU   x'08' in middle of scrolling back
                                    through format code
BKPTR    DS    1C    unused
         DS    0F
DUMMYIRS DS    18C   mostly unused (see below)
         ORG   DUMMYIRS
DUMMYVW  DS    7C    unused
A00017   DS    1C    unused
```

```
DUMMYFL1 DS    H     used   under   name  LENSAVE  in
                                  MAKEBUFF   to   save
                                  total    length   of
                                  multicap field which
                                  splits   over   page
                                  boundary
DUMMYLB1 DS    H     unused
DUMMYFL2 DS    H     unused
DUMMYLB2 DS    H     unused
DYMMYBPT DS    1C    unused
DUMMYFPT DS    1C    unused
         ORG
MAPPTR   DS    F     ->  end of  corrmap generated so
                                  far,    updated   in
                                  MAPROUT.     ENDLINE
                                  sets  up DCMAPLEN in
                                  WCB from it
TRTTAB1  DS    256C  used in MAKEBUFF for looking for
                                  %,x'00',!,caps
VIEWPTR  DS    F     ->  current viewspec string, set
                                  in MAKEBUFF
CSRS     DS    1C    where GETENTRY passes back index
                                  to  new SCB entry to
                                  ORDERS
BUFFPTR  DS    F     to  pass ->  current buffer posn
                                  to    JUSTIFY   from
                                  ENDLINE
PAGEDISP DS    H     used  by  NEWLAB  as  parm  when
                                  calling  GETLAB  (to
                                  pass  back  disp  on
                                  pg)
DELSAVE  DS    F     save  ->  delimiter  reached  in
                                  scanning       text
                                  string.  Needed when
                                  handling   multicaps
                                  over a page boundary
                                  (MAKEBUFF)
LASTPTR  DS    F     unused
OLDVIEW  DS    7C    unused
XTRACHAR DS    X     something   to  do  with  online
                                  underscoring    (not
                                  used/implemented)
                                  (ENDLINE,   NEWLINE,
                                  JUSTIFY)
OLDPTR   DS    F     -> previous first level corr map
                                  entry  for inserting
                                  field   of   disp  to
                                  next   first   level
                                  entry (MAKEMAP1)
BTLPTR   DS    F     ->  BTL  current  entry  (set in
                                  ORDERS, never used)
```

```
SPTR     DS    F     unused
TEMP     DS    1C    temp storage, used in MAKEBUFF
FILENO   DS    H     file number of file currently
                     building buffer
                     from. Set from
                     DBDISP in WCB,
                     updated by NEWLAB if
                     change files in
                     middle
VIEWCNTL DS    7C    contains viewspec string in
                     effect at position
                     in buffer currently
                     being filled in
TOPPTR   DS    1C    unused
TEM2PTR  DS    F     used as temporary storage in
                     MAKEBUFF and ORDERS
SVIEWPTR DS    F     -> old viewspec string on
                     jump/pmuj with
                     viewspec change
                     (ORDERS)
STACKPTR DS    F     unused
OLDFILE  DS    H     old file number when taking
                     jump/pmuj or tag.
                     Set in NEWLAB. Used
                     in ORDERS to set up
                     stack entry
OLDLAB   DS    H     in ORDERS - save int lab on old
                     side of jump/pmuj or
                     tag/annot block for
                     setting stack entry
STARTPTR DS    F     contains no. of chars on line
                     when line is empty.
                     Is zero unless have
                     line numbers.
TEMPARM1 DS    F     used as temp parm in MAKEBUFF,
                     ORDERS
J        DS    F     unused
TEMPVAL  DS    F     used as temp storage in ORDERS
                     (-> end of space for
                     constructing decimal
                     label numbers)
I        DS    F     unused
SAV002   DS    18F   savearea for MOVEPASS, which no
                     longer exists
BUFFMAP  DS    F     parm to MAPROUT routines -
                     current buffer
                     position
DATAMAP  DS    F     parm to MAPROUT routines -
                     current position in
                     file
MAPFLAG  DS          flag parm to MAPROUT
```

```
GENFLAG  EQU   x'80' generated text
JUMPFLAG EQU   x'40' set    by    NEWLAB   indicating
                                non-linear   travel.
                                MAKEMAP2  then  sets
                                corresponding   flag
                                in map entry
FRSTFRST EQU   x'20' on if first 1st level entry
LASTCOR  EQU   x'10' supposed to do something special
                                for   last   char  in
                                buffer    if    line
                                filled       without
                                padding,   but  code
                                missing (ENDLINE)
NOMAP    EQU   x'08' turned  on by  NEWLAB IF *SPLICE
                                CANNOT BE COMPLETED*
                                msg  is used - means
                                ENDLINE  should  end
                                buffer
TBTL     DS    X     temp index  to current BTL entry
                                - if genning buffer,
                                not   scrolling   or
                                typing
PASSBLOK DS    40C   used  in  MOVEPASS,  nonexistent
                                routine now
SAV003   DS    18F   NEWLAB's savearea
LABELPTR DS    F     used  in   NEWLAB  for  call  to
                                GETLAB
LABELDSP DS    F     used  in   NEWLAB  for  call  to
                                GETLAB
TEMPARM2 DS    F     temp parm, used in ORDERS
TEMPARM3 DS    F     temp parm, used in NEWLAB
SAV004   DS    18F   unused
SAV005   DS    18F   VIEWMOVE's savearea
SAV006   DS    18F   MOVELAB, MOVEKEY saveareas
SAV007   DS    18F   unused
TEMPCHAR DS    3C    temp storage,  used in ORDERS to
                                fake   out   NEWLAB.
                                Puts  in intlab from
                                OLDLAB, calls NEWLAB
                                ->  3  before  it as
                                tho   ->   at  order
                                containing    intlab
                                (must   be  halfword
                                aligned)
SAV008   DS    18F   MOVEORDR's savearea
ORDRLEN  DS    H     unused
ORDRTYPE DS    1C    flags used in MOVEORDR
BIT1     EQU   x'80' unused
BIT2     EQU   x'40' opcode displayed
BIT3     EQU   x'20' unused
BIT4     EQU   x'10' showing annotation symbol (|)
```

```
Bit5     EQU   x'08' showing  modifier  bit of splice
                                 or ecilps
ORDHOLDR DS    5C    temp  storage  for  constructing
                                 order in MOVEORDR
SAV009   DS    18F   ENDLINE's  savearea; referred to
                                 by     ORDERS    and
                                 MAPROUT  when faking
                                 end of buffer
SAV010   DS    18F   unused
SAV011   DS    18F   NEWLINE's savearea
TEMPS    DS    0F
TEMP2    DC    f'0'  used   in  ORDERS  for  internal
                                 label  computations,
                                 cleared first
T1       DC    f'0'  unused
PL001    DS    02F   parm  list,  used  in  MAKEBUFF,
                                 ORDERS
PL002    DS    1F    parm  list,  used  in  MOVEPASS,
                                 nonexistent routine
PL003    DS    6F    parm list, used in NEWLAB
BLANKTAB DS    256X  set  up  in  MAKEBUFF,  used  by
                                 ENDLINE to translate
                                 blanks  to something
                                 else     if    being
                                 displayed specially
SAV012   DS    18F   ORDERS' savearea
FSAVE    DS    18F   unused
SPARE001 DS    5F    used   under  name  SKCHNAME  by
                                 ORDERS  for  getting
                                 picture   name  from
                                 NANUPAGE
ONSAVE   EQU   *
SAVEON   DS    18F   used  in ORDERS  as temp storage
                                 for  DECLABELS  from
                                 SCOPE
TMPDCBUF DS    CL112 temp  buff  for constructing dec
                                 labels (ORDERS)
STARTADR DS    F     unused
CODEHEAD DS    F     unused
FSTRTBUF DS    F     unused
FSTRTLIN DS    F     unused
FSTRTMAP DS    F     unused
PACKED   DS    1D    unused
LOCINT   DS    F     unused
LOCEXT   DS    F     unused
MAXEXT   DS    F     unused
FALOCINT DS    F     unused
FLMAX    DS    F     unused
JUSTSAVE DS    2F    workareas and such in JUSTIFY
JUSTABLE DS    20F   workareas and such in JUSTIFY
JUSTWORK DS    34F   workareas and such in JUSTIFY
```

```
ONFLAGS  DS    X     cleared  in MAKEBUFF and ONLINE,
                           never accessed.
DELIMIT  DS    X     unused
INTFLAGS DS    X     cleared   in   MAKEBUFF,   never
                           accessed
PROPCHAR DS    X     unused
FFFLAGS  DS    X     unused
NUMNEED  DS    X     used in JUSTIFY
NUMBLANK DS    X     used in JUSTIFY
SVIEWLEN DS    H     to   hold   length  of  viewspec
                           string     set    by
                           VIEWMOVE,   used  by
                           ORDERS  for  calling
                           VIEWTRAN
```

Use of flag FIRST -

in MAKEBUFF

1) indicates if first time through initialization. Skips some stuff if not.
2) on if string of text is handling starts with capital letter
3) set if are starting display or scroll at start area line

in ORDERS

1) same as #3 of MAKEBUFF

in VIEWMOVE

1) used in putting viewspec string which is longer than 1 display line into buffer

in MOVEKEY

1) used to flag if need initial delimiter or not, i.e., if called from beginning or middle of keyword string.

6 EDITING

6.1 OVERVIEW

Editing is a sub system of FRESS. It comprises a number of commands and has its own mainline. There is a range of opcodes (86 to 143 or x'56' to x'8F') which determines editing commands. When MAINLINE receives a command with this range it calls EDITMAIN to process the command. There are also a few miscellaneous commands for which control is passed to EDITMAIN (Get Label, Return, Jump, Checkfile).

Editing routines are the only routines in the system which modify pages in a FRESS file (other than PAGING). The routines are grouped together for more efficient dynamic storage management. During the course of an edit, the low level routines might be called very often. It would therefore be inefficient to have them call GETMAIN each time they are called. Therefore, editing routines have the following storage allocation scheme.

All editing routines which are called only for editing (the vast majority, the exception will be explained later) are divided up into 8 logical levels, called 0 to 7. A routine on level n may call a routine on level m if and only if m>n. Each level has a certain amount of storage associated with it, equal to the largest amount any routine on that level needs. The amount of storage needed on each level is set by assembling GET13. GET13 contains a macro called LEVELS which defines both the number of levels and the amount of storage on that level. The first level is called level 1. Only EDITMAIN is said to be on level 0.

The macro shows the total amount of storage in EDITSIZE which is an entry point in GET13. EDITMAIN uses IEAFARTS to get that much storage whenever it is called. Then the subordinate editing routines get the storage by calling GET13 and passing it a level number. There are macros to enter and exit editing routines using GET13:

```
<name> REDIT <level>
        •
        •
```

•
REDITRET.

Editing Routine Conventions:

1. R0 is kept at 0 for compares.
2. Error indications are made by putting a number in MERRCODE (in the UCB).

There are six different spaces in which editing may occur: main text, work, annotation, structure, label, and keyword. The main text, work, and annotation spaces contain basically the same kind of data and hence are all handled by the same routines. Any edits done to any structure in these three spaces must be reflected in the structure space, hence, editing in the structure space is handled by the same routines. Thus, whenever any structure is changed, the equivalent change is made in the inverse space too (inverse of structure space is either main text, work, or annotation, and vice versa). Inserts, deletes, substitutes, moves, and copies may be done in these four spaces.

Editing within the label and keyword spaces includes inserts, deletes, and substitutes, and will be covered in more detail later.

From now on, the main text, work, and annotation spaces will be referred to as main spaces. They are the only spaces with normal text in them. There are four main routines which handle editing in main spaces. They are INSERT, DELET, SUBSTITU, and MOVE. There is an entry point to MOVE called COPY.

6.2 EDITMAIN

Functions of EDITMAIN (in order of execution):

1. Call IEAFARTS to get (EDITSIZE) bytes of storage.
2. Check for and handle CK command.
   If CK command:
   - If no first parm, call CHECKFLE and return.
   - If first parm = 'ON', turn on CKFFLAG in MODESWIT; call CHECKFLE.
   - If first parm = 'OFF', turn off CKFFLAG and return.

3. If function has a scope pair of DSPTRS along with it, check the linear order of these in the file (first should be before second). If not OK, generate error code.
4. Look up opcode of function in internal table and call it either directly or via LOADER (if a transient routine).
5. If MERRCODE = 0, handle error message.
6. ELSE if CKFFLAG in MODESWIT is on, call CHECKFLE.
7. SET DSTART if in transcription mode.
8. Free storage and return.

Handling Error Messages:

Error numbers from MERRCODE are looked up in a table in the CSECT EDITERRS. This table specifies for each number from one to (currently) 205:

- whether the number has a message associated with it
- whether or not a REVERT should be done because of the error.

Errors without messages are system errors, cause REVERT, and print the message SYSTEM ERROR XXX - PLEASE SAVE TERMINAL SESSION. A similar message can also be logged on statistics segment.

By printing a message I mean move it into the buffer at GMESSLOC for printing.

6.3 EDITING IN THE MAIN AND STRUCTURE SPACES

6.3.1 INSERT

INSERT:

INSERT has two arguments: the first is a DSPTR of the place to insert after, the second is the text to insert. The text to insert must be processed by SCNINSRT (as must all input in a file). It removes percents, etc., and will be explained later.

BLOCSCAN is called on the insert point, to return information about where the hitpoint is (i.e. is it in a data field, an order, etc.). INSERT then distinguishes between two types of inserts: inserts into regular text, and inserts into data fields. The

insert goes into regular text if the hit point was in regular text. If the hitpoint was a % or opcode, then the text goes after the order hit (unless in the structure space, in which only data field inserts are allowed).

If the hit point is in regular text, everything is simple. GETSPACE is called to open up space on the page for the new text, then the new text is moved in. Then the implied insert point is reset.

If the hit point is in a data field, things are a bit more complicated. GETSPACE is called to open up space within the data field, and error checking must be done to make sure the data field does not become too long. Once the data field is created, all other copies of the new data field must be updated (in the inverse spaces). The new field is copied in detail into the copies of the order. If the data field was a jump keyword field, then the keywords on the page must similarly be updated. If the field was any kind of keyword field, then KEYWORDS must be called to update the KEYWORD space. If the field was a label field then LABEDIT must be called to update the label space. Of course the implied insert point is reset.

6.3.2 DELET

DELET:

DELET takes one argument, the scope of text to delete. A scope consists of 2 DSPTRs. They contain bits which specify whether the scope has yet been determined (it might not have been if a qualifier was used). If a qualifier was used, then SCOPEHIT is called to resolve the scope. (As an aside, SCOPEHIT should actually be called in EDITMAIN, instead of by each routine which uses scopes.)

We now have two DSPTRs which determine the starting and ending point of the delete. BLOCSCAN is called on both DSPTRs.

There are two types of deletes, deletes within a data field and all others. Whenever a data field is changed, it must be reflected in all the proper places (as in INSERT). If an entire field is

deleted, the corresponding bit in the modifier byte must be turned off, and the x'00' removed.

The general scheme for removing text from the file is as follows. The text beyond the text removed is moved upward on the page by calling MOVELOOP. This is referred to as "closing up" the text. Then UPDTDSAR is called. (There was no need for INSERT to call UPDTDSAR because it gets space by calling GETSPACE, which itself calls UPDTDSAR).

For non-data field deletes, the text is scanned for percent signs, either orders or x'6C07's, until the second DSPTR is reached. Each piece scanned over is then closed up as explained above.

Notes on deleting orders or parts thereof:

1. Deleting a block start or end causes deletion of the end or start. If an annotation block, the tag referencing it is deleted too. If a decimal block, at the end of the delete, each tag in the file is retrieved to see if it is a tag referencing any deleted decimal label block. Any found are deleted. The tags to be deleted are kept in a list in DELET's storage. If this list overflows the amount of storage allocated for it, the message INTERNAL LIMIT REACHED - TRY A SMALLER DELETE is printed.
2. Area orders may be deleted only if they are not the first or last in the space.
3. If the entire label field on a point is deleted, so is the point.
4. When a jump/pmuj is deleted, so is its associated pmuj/jump.
5. Deleting the level number on a decimal block causes the decimal modifier bit to be turned off, thus turning the decimal block into a regular block. Any tags referencing it are deleted.
6. Deleting the 'P' on a splice (%SP) or the 'C' on an ecilps (%EC) turns the splice into a jump by turning off the splice modifier bit.
7. Deleting an annotation tag causes the corresponding block start and end to be deleted.

6.3.3 SUBSTITU

SUBSTITU:

SUBSTITU takes two arguments, one scope to delete, and a string to replace it with. It does not in general handle orders. If an order is encountered in scanning the text to delete, INSERT and DELET are called (in that order) to do the substitute. SUBSTITU is in general a straightforward routine and handles all normal substitutes in regular text. It also handles substitutes in label fields. The reason it handles labels is as follows. Consider a substitute on a label if done with INSERT and DELET.

```
original field:  The FRESH system   (1)
edit:  su/H/S
after INSERT:  The FRESSH system   (2)
after DELET:  The FRESS system   (3)
```

The problem is, that although (1) and (3) are less than or equal to 16 chars (i.e., legal), (2) is 17 chars, so INSERT would not allow it.

INSERT could not be called after DELET, because the DELET could possibly delete the whole field (s/cat/dog). Hence SUBSTITU must handle labels.

6.3.4 MOVE

MOVE

MOVE (COPY) is probably the least understood editing routine. It handles all moving of text in the system (move, copy, make annotation). Any structure moved must be reflected in the structure space; furthermore, moves of structure can be done from the structure space. Structure can be moved within a file, but not between files. Structure cannot be copied at all. MOVE can move large chunks of text in two ways: 1) actually moving the text; 2) relinking pages. There have been times when MOVE has linked pages wrong, so this part of the code can be patched out easily. Any orders moved are updated in the internal label space. There is really not much more to say about MOVE except that it is very large, and little understood.

STANDARD (PREDEFINED) MOVES

There are several commands which involve a move or a copy that are not specified using M or CO. They are:

MTL, CTL - move, copy to label

MTW, CTW - move, copy to work space

MFW, CFW - move, copy from work space.

Each of these commands is actually executed by MOVE, but there are three routines which set up parameters for MOVE before calling it.

MCTOLABL handles the MTL, CTL commands, an internal get label is done on the specified label, and the DSPTR for the label (after it) is made the move-to lp for MOVE. Then MOVE or COPY is called.

MCTOWORK handles the MTW, CTW commands. If the workspace does not exist, it is created. The move to lp is set at the bottom of the work space. Then MOVE or COPY is called.

MCFRWORK handles the MFW, CFW commands. If the workspace does not exist, it is created. The move to lp is set at the bottom of the work space. Then MOVE or COPY is called.

MCFRWORK handles the mfw, cfw commands. Here the move to lp is already known, and the scope must be filled in. This scope is set from lps of the top and bottom of the workspace, then MOVE or COPY is called.

6.4 INVERSE EDITING

I have covered the routines which handle edits in the main or structure spaces. Editing can also be done in the LABEL or keyword spaces.

6.4.1 Editing in the Label Space

Inserts, deletes, and substitutes are allowed in the label space. The label edited (only one may be edited at a time) is changed everywhere it appears in the file. When it is noticed by INSERT, DELET, or SUBSTITU that the space of the editing is the label space, LABEDIT is called to do the edit. LABEDIT handles all three edits (insert, delete, substitute) in the label space. The label editing must be updated in the label, main, and structure spaces.

6.4.2 Editing in the Keyword Space

Inserts, deletes, and substitutes of a single keyword are allowed in the keyword space. When it is noticed by INSERT, DELET, or SUBSTITU that the space of the edit is the keyword space, KEYEDIT is called to do the edit. KEYEDIT handles all three edits in the keyword space. The keyword edited is updated everyplace it appears in the file (it can appear on many blocks, jumps, etc.).

Note: There is code in KEYEDIT to handle another kind of edit in the keyword spaces, but it is currently no-oped because it was poorly designed and had major bugs in it. Also, the feature is of little use. Example:

```
keyword          4%<     3%J     3%P     8%T
```

If "%T" is deleted, then the keyword is to be removed from all tags it is on. This is the feature which is no-opped currently. The code is still there though, and could be debugged.

This concludes the discussion of normal editing in all spaces.

6.5 MAKING STRUCTURE

The general scheme for adding an order to a file is as follows:

1. Call GETSPACE to get enough room in the file at the desired point for the order and data fields (if any).

2. Call GETINTLB to get the next internal label in the file and indicate in the lab space what kind of order it is and what page it is on.

3. Make the order, put in the %, the opcode, the modifier byte, and intlab.

4. Put the data fields after the order, followed by x'00', in the order in which their bits appear in the modifier byte.

5. Reflect any new labels or keywords in the corresponding space.

6. Put a copy of the order in the correct place in the structure space. This is done as follows. The place where the order goes is an lp, and BLOCSCAN has been called on that lp.
   BLOCSCAN returns (among other things) the internal label of the last piece of structure before the lp. It gets this from the header at the top of the page, and by scanning down the page till it gets to the lp. Once we know this intlab, we call GETDECLB to get a pointer to it in the structure (used to be decimal label) space. Then put a copy of the order after the order in the structure space.

7. Call SCNENDPG. As I have mentioned, the intlab of the last piece of structure previous to a page is stored in the header of that page. SCNENDPG is the routine which updates this field. When we put an order on a page, that may or may not affect the intlab on the next page. SCNENDPG is called on the page on which we made the code, and it updates correctly. See the description of SCNENDPG for more information.

6.5.1 MAKELABL

MAKELABL handles the MLABEL command and has two functions.

1) to create a point with a label field in it

2) to add a label field to an existing block

If the lp at which to make the label is a block start without a label field, then a label field is added to the block. Otherwise a point order is made to hold the label field.

6.5.2 MAKEAREA

MAKEAREA handles the NAREA command and makes a new area at the very bottom of the file. An area order is a tag with a modifier byte of x'90': x'6C8190intlab' .

6.5.3 SPLTAREA

SPLTAREA adds an area at the specified place in the file, i.e., it handles the SPLITAREA command.

6.5.4 MAKEBLOK

MAKEBLOK handles (in full or in part) the following commands: MBLOCK, MDBLOCK, IBLOCK, IDBLOCK.

If doing an insert block of some kind, MAKEBLOK makes a block start and end at a given lp. Very simple.

If doing a make block of some kind, things are not quite so simple. A block start and block end must be placed around a specified scope, but block nesting must be checked for. For example, consider a file which contained

...%<..A..%>..B..

A block cannot be made from A to B, because the block would contain an unmatched block end. The check for this is done by counting blocks in the structure space. Start at A, say, with a counter at zero. Scan for blocks and area codes until you get to B. If an area code is found, the block is illegal. When you

hit a block start, add 1 to the counter. When you hit a block end, subtract 1. When you reach B, the block will be nested properly if the counter is zero. For decimal blocks, the correct bit in the modifier byte is turned on.

6.5.5 ANNOT

ANNOT handles the commands IANNOTATION, RTANNOTATION, MANNOTATION.

- For IA,, a block is made at the bottom of the annotation space (the space is created if necessary), and a tag referencing the block is made at the specified lp. If keywords are specified, they are put on the tag and the block start. Then the specified text is put in the block.

- For MA, a block is made at the bottom of the annotation spaces and a tag referencing it is placed after the specified scope. Then MOVE is called to move the specified scope into the block.

- For RTA a block is made around the specified annotation block, and a tag referencing it is placed at the specified lp.

All blocks made are annotation blocks.

6.5.6 MAKETAG

MAKETAG handles the commands MDREF, MDRDEF, MPREF.

- for MDR and MDRD, a tag is made at the specified lp which references the specified decimal label block. The internal label of the block referenced is the data field on the tag.

- for MPR, a tag is made at the specified lp which references the specified picture. The picture number is the data field on the tag.

6.5.7 MAKEJUMP

MAKEJUMP handles MJUMP and MSPLICE commands. The only difference is that for MS the splice modifier bit is turned on.

A jump (splice) is made at the specified lp with the specified keywords, explainers, and viewspecs, and a pmuj is made at the other lp.

The only tricky part is if the 2 lp's are in different files, then DSLINK is used to correlate the 2 files' internal labels.

6.6 INPUT MODE

There are a number of ways FRESS can enter input mode (other than offline read):

1. By typing "i", in which case CLINTERP turns on the flag INPTMODE in MODEFLAG, then CLINTERP continues reading lines in input mode. "INPUT" is typed by the low level attention routine upon request from ATTEND.

2. By specifying null text to insert on an Insert or Substitute - if Insert, INSERT just sets the implied insert point to the specified lp, sets INPTMODE, and returns. It also sets the flag for ATTNxxxx to type INPUT. Each line typed in input mode goes into the file at the implied insert point.
   If Substitute, the specified text is deleted, then INPTMODE is set, and the implied insert pointer is set.

3. By specifying null text on an "IA", ANNOT makes the annotation tag and block, sets the implied insert point in the block, and sets INPTMODE.

4. By specifying BI or TI. Both commands are handled by TBINPUT. TBINPUT just sets the implied insert point at the appropriate place (top or bottom of file), and sets INPTMODE.

5. By specifying no text in IB or IDB. Both commands are handled by IBLOCKS. First MAKEBLOK

is called to make the block at the given place and set the implied insert point in the block, then IBLOCKS calls INSERT if there was text, otherwise INPTMODE is set.

When INPTMODE is set, CLINTERP interprets each line as the text to insert at the implied insert point. For example, typing "dog" in input mode is the same as typing "i/dog" in non-input mode.

6.7 SURROUNDING FUNCTIONS

There are a number of editing commands which have the effect of surrounding a scope of text with other text. The routine which actually does this surrounding is INAROUND. It takes as arguments the scope and two strings of text, one to put in front of the scope, one to put after it. INAROUND is called by SURRMAIN, which sets up the correct arguments according to the command. Commands handled by SURRMAIN are:

COMMAND:	INSERTS:
surr/scope/t1/t2	t1 & t2
u/scope	!(0 & !)
mfont/scope/n	!(n & !)
rbars/scope	!-r- & !-r-
footnote/scope/n	!-fn- & !-f-

SURRMAIN checks the validity of any text to insert (i.e., calling SCNINSRT.) INAROUND just inserts the text. If either end of the scope is in an order, then the text is inserted after the order. This is a good routine to look at to see how basic inserting is done. It handles only the simple case, unlike INSERT.

6.8 CHARACTER CASE COMMANDS

There are three editing commands to change the case of a scope of text: Capitalize, Uncapitalize, and Fcase (flipcase). All three of these commands are handled by CAPIT. CAPIT takes each character in the given scope (ignoring orders) and does the appropriate change. In the case of UNC and FC, the first non-blank character after certain punctuation markers is not changed.

6.9 CREATING AND DELETING PAGES IN A FILE

Pages are created by calling the paging routine CREATE. Simple. It is a little more difficult to delete pages. To delete a page, assuming it has been linked out of any chains,

1) call DELETPG, a paging routine to delete a page

2) UPDTDSAR must be notified whenever a page is deleted.

6.10 UPDATING DSPTRS

During the course of an edit there can be many DSPTRs identifying places in the field. They can come from stacked (>) commands, the implied insert point, the implied insert stack, the DSPTRs of the current edit, the return stack, the memory ring, etc. Changes made to the text within a file may or may not affect the DSPTRs. If it does, the DSPTR must be updated. For example, consider DSPTRs indicated by A, B, and C below. Maybe C is the implied insert point. Anyway A and B denote a scope to delete. Once the delete is done, all the text starting at B will be moved up the page to A, hence C will move. DSPTR C must be updated to reflect the change. This is one of many examples.

This updating of DSPTRs is done by UPDTDSAR (for update data structure). UPDTDSAR is given information about what kind of change was made, then it searches for every DSPTR and checks to see if the change affects them. UPDTDSAR has many, many modes of call, too many to

list here. The calls can be found in the various editing routines. Briefly, UPDTDSAR must be called when:

- deleting text on a page
- deleting an entire page
- inserting text onto a page
- moving text from one page to another (so as to balance pages, or a move), etc.

6.11 KEEPING PAGES BALANCED

There are a number of mechanisms for keeping pages in a file balanced, i.e., not having a page with x'600' bytes on it while the next page has x'10', etc. There are two methods used, one when adding text to the file, the other when deleting or moving text.

6.11.1 GETSPACE, SPLIT

Whenever space on a page is needed for inserting, GETSPACE is called to get that amount of space. If there is room on the page for the specified amount of text, then the text below the insert point is moved down and UPDTdSAR is called by GETSPACE to reflect this change.

If there is not enough room on the page for the text, then GETSPACE calls SPLIT to "split" the page up into pieces. SPLIT tries three schemes: 1) it checks to see if the text on the current and previous pages can be better balanced to allow for the space; 2) it does the same thing for the current and next pages; 3) it creates a new page and puts some of the text on the current page on it. When text is moved from page to page, a) UPDTDSAR must be called, and b) any orders in the text moved must have their internal labels updated in the intlab space.

Note: Currently (as of 5-24-74), schemes 1 and 2 of SPLIT are patched out, because they contain a bug which involves updating the intlab space incorrectly. They are not too useful, and the system runs fine without them.

6.11.2 PAGEBAL

DELET and MOVE both call a routine called PAGEBAL to try to keep pages balanced. PAGEBAL should be called when text has been moved or deleted from a page. PAGEBAL is passed a pointer to the page. PAGEBAL tries to fit the text on the passed page on either the previous or next page, depending on a parameter passed. If it can, a page will be deleted from the file.

6.12 MOVING ORDERS INTERNALLY

Care must be taken when moving text from page to page that an order is not split between two pages - each order must stay on one page. The following scheme shows how this is avoided. Say (doing a SPLIT), you wanted to move half the text on the following page to a newly created page: You calculate the midpoint of the text on the page. You then call BLOCSCAN on this point, and if m is in an order, BLOCSCAN will return you a pointer to either before or after the order (depending on a flag passed), so if there is an order starting at B and ending at C, then BLOCSCAN might return C as the adjusted midpoint, and the text from C down could be moved to a new page.

6.13 TEXT SCANNING ROUTINES

There are two routines for scanning text input to a field. Text in a file may not contain x'6C's or x'00's . All text to enter the file (entered by the user) must pass through SCNINSRT. SCNINSRT scans the text passed for tab key hits, turning them into "!-T-". It also handles capitalization via % and %%. Finally, it interprets backspaces and underscores, turning, for example, (<-_b_<-<-_ c" into !(0abc!), or "bbb<-<-<-///" into "b<-/b<-/b<-/", as is required by the FU program.

All keyword strings or parts thereof must also pass through DEBLANK, which removes blanks around delimiters in keyword strings, i.e., "key 1 | key 2", becomes "key1;key2"

6.14 RESOLVING SCOPE QUALIFIERS

The following scope qualifiers are currently supported:

-l = line
-w = word
-b = block
-o = order
-c = character

The line qualifier is a special one, and is handled before editing is called. The others, however, are handled by editing. The first byte of each DSPTR is a byte of flags telling whether it is a single DSPTR, a pair (scope), or a qualified scope. If a qualified scope, SCOPEHIT is called to handle the qualified scope. SCOPEHIT is called by every editing routine which handles scopes.

The w and c scope are fairly self-explanatory. If the 0 qualifier is used, the lp must be part of an order, and if so, the scope is set to be the % of the order. If the block qualifier is used, the lp must be somewhere within a block. The resulting scope will be the block start to the block end in the main text space. If the lp was in the structure space, however, and the command was a MOVE, pointers in the structure space are returned.

6.15 UPDATING AND SCANNING SYSTEM SPACES

6.15.1 Keyword Space

There are two editing routines which maintain the keyword space. FINDKEY scans it for keywords, but never changes it. KEYWORDS does all updating in the space. Keywords is passed on old and new keyword string whenever keywords on an order are changed. It updates the keyword space accordingly. KEYWORDS has its own GETSPACE routine internal to it, and is hence the only routine ever to change the keyword space.

6.15.2 Label Space

There are two editing routines which maintain the label space. FNDLABEL bisects the space for a given label, and EDTLABEL updates the space. EDTLABEL is passed on old and new labels and makes the appropriate changes in the space.

6.15.3 Dslink Space

DSLINK allocates entries in the dslink space. It is called only by MAKEJUMP currently. DSLNKSCN scans the space for a given internal label. Entries in the space are deleted by DELET.

6.15.4 Internal Label Space

GETINTLB looks up a given intlab in the intlab space. The actual updating of the entry returned is done by whoever called GETINTLB. GETINTLB can also be called to allocate the next internal label, by calling it to look up intlab x'0000'.

6.16 RETRIEVING ORDERS BY INTERNAL LABEL

Given the internal label of an order, it is possible to get a pointer to the order, in either the main or structure spaces. GETLAB returns the order in the main space as follows. It first looks up the intlab in the intlab space. The entry tells the page the order is on. Then that page is gotten and scanned until the order is found. GETDECLB returns the order in the structure (used to be decimal label, hence the name) space. It does this by scanning the structure space from the beginning of the order.

6.17 MISCELLANEOUS EDITING ROUTINES

6.17.1 FIX

The routine FIX executes the commmand FIX, which is a debugging command. FIX has two functions:

1. to recreate the structure, internal label, and keyword spaces (or any combination thereof) from the main text space of a file.
2. to convert an old system file to version 6.0.

An old version file is identified by PCBGUYID (in FCB) being H'O'. In a new file, the next internal label is kept in PCBGUYID. When a file is opened, if PCBGUYID = 0, then FIX is called to do the conversion. This test is valid only for a file created by the CMS version of FRESS. This field must be zeroed manually for files which have been converted from the MVT version in order to convert them for version 6.0 using FIX.

FIX does slightly different things depending on whether it is fixing a file or converting one (it tells by checking PCBGUYID).

If it is fixing a file, then it reads a line from the terminal, expecting a line with any combination of (I,S,K) on it to tell what spaces to fix. Then, if it is going to fix the structure or keyword spaces, it deletes them. It then scans the main, annotation, and work spaces looking for structure. When it finds an order,it:

1. updates intlab space entry if necessary
2. adds order to structure space if necessary
3. adds keywords to keyword space if necessary

Also, it zeroes the header field LASTINTP.

Then SCNENDPG is called on main, annotation, and workspaces, to update header information.

This will fix any blown structure, intlab, or keyword spaces assuming the main text space is okay (it usually is).

When converting a file, basically the same stuff is done (ISK is assumed) with the following exceptions:

1. PCBGUYID must be set.

2. each page header must be converted from old to new format, the new format is smaller, so all the text on the page is moved up.

3. If any jumps are found, they are overlaid with *'s, etc.

6.17.2 DPICT

The routine DPICT executes the DP command, and deletes a picture from the picture space. The picture space is maintained separately from editing, but DPICT calls DELET to delete any tags referencing the deleted picture.

6.17.3 MOVELOOP

There is a routine in FRESS to do any arbitrary length MVC. It is called MOVELOOP. It is used by editing to move almost all text MVC'd.

6.17.4 ENDORDER, ENDORD

It is often desired to get to the end of an order you are pointed at (for instance "i/%=(label)/text" puts the text after the label, but the lp inserts receives the 6C). In versions of FRESS before 6.0, this was a harder thing to do than it is now. Therefore, ENDORDER is called to get over the order. ENDORD was an entry point which at one time did something slightly different than ENDORDER, but now both are equivalent, and simply TRT over any and all data fields of the order.

6.17.5 CHECKFLE

CHECKFILE executes the CK (check file) command, and checks the integrity of the PCB dictionary and the FREEAREA of each page.

6.17.6 BLOCSCAN

BLOCSCAN is called, given a pointer into a page, and returns a lot of information about that point in the text. The information it returns is shown in the SCOPE macro. It returns information such as:

- last intlab before the given point
- area number
- if an order:
    - opcode - x'80'
    - ptr to start of order
    - if a data field
        - ptr to start of data field

etc.

7 MISCELLANEOUS COMMAND HANDLING

8 CMS DEPENDENCIES

TABLE OF CONTENTS