AD 731348

ARPA ORDER NO.: 189-1

R-603-ARPA

August 1971

The ISPL Basic File System and File Subsystem for Support of Computing Research

E. F. Harslem and J. F. Heafner

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

46

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0141. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of ARPA.

DOCUMENT CONTROL DATA

1. ORIGINATING ACTIVITY The Rand Corporation	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP
3. REPORT TITLE THE ISPL BASIC FILE SYSTEM AND FILE SUBSYSTEM FOR SUPPORT OF COMPUTING RESEARCH	
4. AUTHOR(S) (Last name, first name, initial) Harslem, E. F., J. F. Heafner	

5. REPORT DATE August 1971	6a. TOTAL NO. OF PAGES 48	6b. NO. OF REFS. -
7. CONTRACT OR GRANT NO. DAHC15 67 C 0141	8. ORIGINATOR'S REPORT NO. R-603-ARPA	
9a. AVAILABILITY/LIMITATION NOTICES DDC-A	9b. SPONSORING AGENCY Advanced Research Projects Agency	

10. ABSTRACT

Functional specification of the Basic File System and one of several file subsystems envisioned for the ISPL computing system ~~described in R-562~~. A generalization of the GRAIL logical input/output system ~~(see RM-6257)~~, the software package is described in its ISPL context. The BFS, as a resident part of the ISPL Operating System, creates, modifies, deletes, and keeps directories of file subsystems, allocates their secondary storage space, and handles their I/O transmissions. BFS is unconcerned with file structure, which is separately determined by each FSS. Any console or program can address any file by giving its name, FSS, and a qualifier. Further communication is determined by the rules of that particular FSS. All BFS procedures report whether and why an operation failed. An ARPA research file, FSS1, is described in its temporary IBM 360 implementation, with indications of probable changes in the ISPL environment.

11. KEY WORDS

File Structure and Management
Computers
ISPL
GRAIL

ARPA ORDER NO.: 189-1

R-603-ARPA

August 1971

The ISPL Basic File System and File Subsystem for Support of Computing Research

E. F. Harslem and J. F. Heafner

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

PREFACE

This Report is a functional specification of software development for the Incremental System Programming Language (ISPL) Operating System and support programs. The work was sponsored by the Department of Defense's Advanced Research Projects Agency (ARPA) and is an integral part of both Rand's and the client's overall program to explore current computer technology. Specifically, the work is directed toward providing an ISPL to be used as a tool for further computer-based research.

This Report describes the Basic File System (BFS) and one of several envisioned file subsystems for the ISPL machine. The primary audience for this Report is the software development group and others responsible for the design and use of the ISPL Operating System. Therefore, the reader is assumed to be familiar with the companion documents: R. M. Balzer, *The ISPL Machine: Principles of Operation*, The Rand Corporation, R-562-ARPA, August 1971; and R. M. Balzer, *The ISPL Language Specifications*, The Rand Corporation, R-563-ARPA, August 1971.

Although the file organization presented here was generated as a specification for ISPL, its wider applicability should be noted. It is consistent with, but not bound to, the ISPL philosophy.

SUMMARY

This Report is a functional specification of a file system and subsystem. It was provided as a segment of a proposed computing research environment [1,2]. These specifications are a generalization of an earlier research tool [3] and as such can be considered an independent package for file organization and management. However, the specifications are described in terms of an initial version of the Incremental System Programming Language (ISPL) machine assemblage.

The Basic File System (BFS), which is a special file subsystem (FSS), couples the hardware to other FSSs. It is responsible for the actual execution of input/output (I/O) transmission and also for the allocation of secondary storage space to other FSSs.

FSSs manage the actual files. They can be user-written to organize and access files in a way most amenable to processing by user application programs. The particular FSS described here was included to satisfy the needs of two projects sponsored by the Advanced Research Projects Agency (ARPA).

Thus, the organization of a file is directly determined by the FSS used. To assist in processing, FSSs will typically describe the file's contents by some structural or tabular header information. The BFS guarantees that secondary storage space allocated to one FSS will not be accessed by another. Beyond that, the security and privacy of a file must be guarded by the FSS under which it operates.

The BFS and subsystems communicate secondary storage addresses via file pointers (FPs). The FP is a new ISPL data type that was not defined in Refs. 1 and 2.

This Report describes the BFS procedures that have been implemented in ISPL. These procedures allow 1) the

creation, modification, and deletion of subsystems; and 2) secondary storage allocation to subsystems. They also support application program transactions with a subsystem by maintaining directories of information about files for the subsystems.

The FSS described here provides direct and sequential access to file data. It permits scattered read and write operations of variable-length data--a fundamental necessity for some planned application programs. The FSS either retains the secondary storage location (FP) of all data in a file or allows the application program to record this information. Allowing the application program to do this bookkeeping gives the application program greater flexibility in structuring a file to meet user requirements. The subsystem procedures allow application programs to create, destroy, and access the data in a file.

This Report gives the detailed specifications of both the BFS and subsystem's secondary storage organization. Each record type is diagrammed, along with commertary on its use.

Again, this file management system is a research tool; access methods to support production programs, such as those typically written in COBOL, are beyond the intended purview.

ACKNOWLEDGMENTS

The authors thank Rand colleagues Robert M. Balzer, J. Cliff Shaw, and William L. Sibley for contributing to these specifications.

CONTENTS

FIGURES

I. INTRODUCTION

This Report describes a Basic File System (BFS) and a file subsystem (FSS). The BFS is a unique entity within the environment in which it operates, i.e., it has no brother systems performing similar functions, nor is it superimposed on a more basic secondary storage access method. The FSS described is one of many subsystems, each supporting a different form of file use.

This two-level hierarchy of file management (system and subsystem) is intended for implementation on a "raw machine." That is, the BFS should *directly* cause performance of the appropriate file I/O operation. The implementation of this hierarchy "on top" of an existing file system might introduce an unacceptable overhead.

The specifications in this Report are oriented toward an initial, temporary version of this implementation on an IBM 360 using IBM 2314 disk drives for file storage. Thus, a good deal of the nomenclature and description (but not the design philosophy) leans toward this configuration. Therefore, this initial description does not take into account the parallel processing notation of the Incremental System Programming Language (ISPL) [1-2]. Aesthetically the details suggested for implementation should be passed over and attention given to file structuring and managing techniques.

This file organization is not intended to support production systems with production goals. Rather, it is designed to support interactive experimental programming systems. Evidence of the viability of this approach to file management is the Rand-developed GRAIL System [3].

It should be noted that the BFS leans toward distributed rather than centralized knowledge of files--their existence, location, and structure. The BFS merely allocates space to FSSs and performs requested I/O functions.

From the standpoint of the user, the FSSs are completely responsible for file creation and manipulation. This allows great flexibility in each FSS so that they can be tailored to the specific needs of their individual applications.

II. OVERVIEW

CAVEAT

These notes pertain primarily to the Basic File System (BFS) used with the initial ISPL implementation. Indicated are those BFS procedures that should be transferable to the ISPL machine, those that should be made obsolete by the ISPL port mechanism, and the ISPL subroutine calls that will be replaced by language constructs.

FILES

The ISPL file system permits users to create files, name them, store data in them, recall data from them, and save them for later use. All files are catalogued files; i.e., with respect to the BFS they are permanent and must be explicitly purged by the user through a file subsystem (FSS). A file is a collection of data whose gross internal organization is known to a particular FSS, and whose more detailed intra-organization is known only to the user and his procedures. Unlike conventional file systems, the BFS has no knowledge of the file's organization.

THE BFS

The BFS is a resident part of the ISPL Operating System (OS). It 1) invokes FSS procedures to carry out user file requests (this may also be done by the OS); 2) maintains the necessary directories of file and FSS information; and 3) allocates secondary storage for further suballocation for files by FSSs.

The BFS consists of 1) the nucleus of a particular FSS that maintains a file containing directories of file data, matrices of available and assigned track space, etc., and 2) a set of assembler-coded subroutines for channel program construction and I/O transmission management.

FSS

An FSS is a collection of seven primary procedures (see p. 9), declared to the BFS, that process user file operations. An FSS is characterized by a high-level data-structure organization, perhaps known to its users, superimposed upon the intra-organization of all files operating under that FSS. It is further characterized by the kinds of operations it affords its users. FSSs are not part of the BFS nor of the OS.

INITIAL IMPLEMENTATION OF THE BFS ON AN INTERMEDIATE MACHINE

The physical I/O routines are ISPL-callable subroutines written in assembler language with coding conventions closely related to the standards for the intermediate machine.

The remainder of the BFS is a collection of ISPL procedures. The user's interface with BFS is through ISPL procedure calls, assignment statements, and ISPL language constructs. All BFS procedures have one parameter that indicates whether the operation succeeded or failed, and if it failed, why. Unless otherwise stated, all input parameters are treated by the procedures as read-only.

FILE SECURITY AND PRIVACY

The requirements for a private and secure schema for files vary with an individual's needs for each of his files. The BFS circumvents this complex problem by simply relegating such authority and responsibility to FSSs. The BFS allows the console or program to communicate with an FSS if the console or program supplies a valid triplet of file-name, qualifier,† and FSS-name. Further scrutiny of the user and discrimination of his access privileges to the file are determined by FSS/user protocol.

---

†The qualifier is an eight-byte field whose contents are legislated external to the OS. It has been suggested that it represent man number and project number where either may be null.

FILE POINTER

A file pointer (FP) is one of the ISPL pointer types. It can be translated by the BFS to an absolute secondary storage address.

There will be a different FP type for each secondary storage medium. We discuss FPs for IBM 2314 disk drive storage.

There are two candidates for the complete format of this FP. An FP may be a four-byte pointer that addresses a second four-byte file data item, shown below. The alternative is an eight-byte FP that includes the four-byte file data item.

Pack Number	Cylinder Number	Track Number	Record Number

File Data Item of an FP

III. BASIC FILE SYSTEM (BFS) PROCEDURES

PROCEDURES FOR ESTABLISHING AND MAINTAINING A FILE SUBSYSTEM (FSS)

The following procedures allow a program to create, destroy, and update an FSS. The program's interface is initially through the ISPL call statement. Declaration statements will be provided in the language for the ISPL machine. These procedures should transcend the move to the ISPL machine.

1) Function: Institute a new FSS.
   Form: CALL BFSCFSS (PASSWD, SUBSYS, ADDRS, IND)
   Inputs: PASSWD and SUBSYS are ISPL varying character strings of maximum length 20. They specify a password and the FSS name, respectively. ADDRS is a location in an ISPL record containing an ordered list of seven primary procedure addresses (or load-module names) for user file operations.
   Outputs: IND is an ISPL discrete valued variable used as a success/fail (S/F) indicator.

2) Function: Purge an FSS.
   Form: CALL BFSDFSS (SUBSYS, PASSWD, IND)
   Inputs: SUBSYS and PASSWD have the same meaning as in 1), above.
   Outputs: IND is the S/F indicator.

3) Function: Replace a component (one of the seven procedures) of an FSS.
   Form: CALL BFSREP (SUBSYS, PASSWD, ADDRS, IND)
   Inputs: SUBSYS and PASSWD have the same meaning as in 1) and 2), above. ADDRS points to the new component, followed by an

	integer specifying its position in the ordered list.
Outputs:	IND is the S/F indicator.

PROCEDURES FOR AN FSS TO OBTAIN AND RELEASE SECONDARY STORAGE SPACE FROM THE BFS

The BFS allocates space to an FSS, which in turn provides space for files operating under it. Space allocation and file creation are independent as far as the BFS is concerned. File purge and space release to the BFS are also independent with respect to the BFS.

The BFS procedures listed below allocate and de-allocate secondary storage space to an FSS. The basic unit of space in these transactions is a track. The FSS interface is the ISPL call statement. The calls will later be superseded by ISPL resource-allocation constructs similar to those extant for primary storage. These procedures should survive the ISPL machine transition.

1)	Function:	Request currently mounted pack IDs and their residency status.
	Form:	CALL BFSPACKS (P, IND)
	Inputs:	None.
	Outputs:	P contains a list of the following pairs: fixed-length character string specifying pack ID, and discrete valued variable indicating residency status. IND is a discrete valued variable used as an S/F indicator.
2)	Function:	Request available secondary storage space to be assigned.
	Option A	Assign consecutive tracks within a cylinder.
	Form:	CALL BFSASSGN (OPTION, NAME, IDENT, AMNT, F, IND)

Inputs:	OPTION is a discrete valued variable specifying the option code. NAME is a varying character string of maximum length 20 that gives the FSS name. IDENT is a fixed-length character string specifying the pack ID. AMNT is a half-word integer specifying the number of tracks.
Outputs:	F is an FP, i.e., the starting track address. IND is the S/F indicator.
Option B	Assign consecutive tracks.
Form:	Same as option A.
Inputs:	Same as option A.
Outputs:	Same as option A.
Option C	Assign consecutive cylinders.
Form:	Same as option A.
Inputs:	Same as option A.
Outputs:	Same as option A.
Option D	Assign tracks.
Form:	Same as option A.
Inputs:	Same as option A.
Outputs:	F is a pointer to an ISPL record that contains an integer count of "pairs," followed by the pairs. A pair consists of a fixed-length character string specifying pack ID and an FP.
Note:	If all tracks cannot be assigned on the specified requested pack, the overflow is assigned on a resident pack.
Option E	Assign consecutive tracks "horizontally."
Form:	Same as option A.
Inputs:	Same as option A.
Outputs:	Same as option A.

3) Function: Release assigned tracks to the BFS.
   Form: CALL BFSRETN (NAME, IDENT, AMNT, F, IND)
   Inputs: Same as the correspondingly labeled arguments in 2), above.
   Outputs: IND is the S/F indicator.

FSS PROCEDURES FOR USER FILE MANIPULATION

The following procedures support functions that an FSS makes available to users. These FSS functions are further described below (see pp. 18-20); however, they are included here to illuminate the discussion in this section.

The procedures listed below allow users to access files. They will not survive the definition of the port mechanism. ISPL will be extended to encompass multiplexing over ports and other primitive port operations.

These operations may logically be thought of as invoking the BFS or the Operating System (OS). They will probably be implemented so that CREATE and CONNECT invoke the OS; the other operations will then directly invoke the appropriate FSS procedure.

1) Function: Create a file.
   Form: CREATE FILE (P→A, P→B, C, P→D, P→E, IND)
   Inputs: A contains a file qualifier whose contents have not been specified. They will be specified in terms of operational requirements, i.e., individuals and projects. B is a varying character string, of maximum length 20, specifying a file name. C is a varying character string, of maximum length 20, specifying an FSS name. D is a parameter list interpreted by the FSS.
   Outputs: E is a parameter list interpreted by the user. IND is a discrete valued variable used as an S/F indicator.

2)	Function:	Destroy a file.
	Form:	DESTROY FILE (P→A, P→B, P→D, P→E, IND)
	Inputs:	Same as 1), above, with the exclusion of C.
	Outputs:	Same as 1), above.
3)	Function:	Connect a "port" to a file.
	Form:	CONNECT FILE (P→A, P→B, C, P→D, P→E, IND)
	Inputs:	Same as 1), above.
	Outputs:	Same as 1), above.
4)	Function:	Disconnect a file from a "port."
	Form:	DISCONNECT FILE (P→A, P→B, P→D, P→E, IND)
	Inputs:	Same as 2), above.
	Outputs:	Same as 1), above.
5)	Function:	Send to a file.
	Form:	FILE (P→A, P→B, P→D, IND) = Q. Q is an ISPL record.
	Inputs:	Same as 2), above, with the omission of E.
	Outputs:	IND is the S/F indicator.
6)	Function:	Receive from a file.
	Form:	Q = FILE (P→A, P→B, P→D, IND). Q is an ISPL record.
	Inputs:	D is a parameter list interpreted by the FSS. A and B are the same as in 2), above.
	Outputs:	D will also receive the file data. IND is the S/F indicator.
7)	Function:	Communicate with FSS.
	Form:	COMMUNICATE (P→A, P→B, P→D, P→E, IND)
	Inputs:	Same as 2), above.
	Outputs:	Same as 1), above.
	Note:	This procedure permits communications between the user and an FSS for queries, error reporting, etc.

BFS PROCEDURES THAT ALLOW AN FSS TO IMPLEMENT USER FILE OPERATIONS

The BFS procedures listed below are called from an FSS by the ISPL call statement. They should partially survive over the port mechanism definition.

The procedures outlined here are not in one-to-one correspondence with the procedures listed above for user file manipulations.

1) Function: Create a file index entry.

Note: The BFS maintains a directory of file information whose entries are created by this procedure. Recall, however, that only an FSS creates a file or knows its actual location.

Form: CALL BFSCFIE (QUAL, NAME, SUBSYS, PFC, PFPTR, RDFP, IND)

Inputs: QUAL contains a file qualifier whose contents have not been specified. They will be specified in terms of operational requirements, i.e., individuals and projects. NAME is a varying character string, of maximum length 20, specifying a file name. SUBSYS is a varying character string, of maximum length 20, specifying an FSS name. PFC is a variable-length field (varying character) whose content is known only to the FSS. PFPTR is a half-word integer giving the length of the PFC field.

Outputs: RDFP is an FP for later accessing the FSS field. IND is a discrete valued variable used as an S/F indicator.

2)	Function:	Delete a file index entry.
	Form:	CALL BFSDFIE (QUAL, NAME, SUBSYS, F, IND)
	Inputs:	QUAL, NAME, F, and SUBSYS have the same meanings as in 1), above.
	Outputs:	IND is the S/F indicator.
3)	Function:	Access an FSS field.
	Form:	CALL BFSGETPF (QUAL, NAME, SUBSYS, PFC, RDFP, IND)
	Inputs:	QUAL, NAME, SUBSYS, and RDFP have the same meanings as in 1), above. RDFP is an option, but providing the FP saves secondary storage read and search time.
	Outputs:	PFC and IND have the same meanings as in 1), above.
4)	Function:	Replace an FSS field.
	Form:	CALL BFSPUTPF (QUAL, NAME, SUBSYS, PFC, RDFP, IND)
	Inputs:	QUAL, NAME, SUBSYS, PFC, and RDFP have the same meanings as in 1), above. Again, RDFP is an option.
	Outputs:	IND is the S/F indicator.
5)	Function:	Change a file name and/or qualifier.
	Form:	CALL BFSMODFY (OLDQUAL, OLDNAME, SUBSYS, NEWQUAL, NEWNAME, IND)
	Inputs:	OLDQUAL, OLDNAME, and SUBSYS have the same meanings as in 1), above. NEWNAME is a varying character string of maximum length 20 specifying a new file name. NEWQUAL is an optional new qualifier.
	Outputs:	IND is the S/F indicator.

6)	Function:	Write data in file.
	Form:	CALL BFSWRITE (SUBSYS, I, RDFP, COUNT, KEY, IND)
	Inputs:	SUBSYS and RDFP have the same meanings described in 1), above. I is an ISPL pointer followed by an integer count of ISPL descriptors. The descriptors consist of a pointer and an integer length. COUNT and KEY are optional; they correspond to a track's count and key fields, respectively.
	Outputs:	IND is the S/F indicator.
	Note:	An initial restriction is that only a single physical record on a single track may be written for each instance of the procedure. Note that I provides for scatter write from primary storage.
7)	Function:	Read data from file.
	Form:	CALL BFSREAD (L, RDFP, COUNT, KEY, IND)
	Inputs:	RDFP is the same as described in 1), above. L contains a pointer and an integer length for the read.
	Outputs:	L indicates the location and amount of data to be received. COUNT and KEY have the same meanings as in 6), above, and are optional. IND is the S/F indicator.
	Note:	An initial restriction is that only a single physical record on a single track may be read for each instance of the procedure.
8)	Function:	Query file information.
	Form:	CALL BFSQUERY (QUAL, NAME, SUBSYS, M, RDFP, INDEX, IND)

Inputs: QUAL is a file qualifier whose left or right half may be ignored as a function of INDEX. SUBSYS is an FSS name. INDEX is a discrete valued variable used as an index to specify the search options, which are:

a) Match on left half of qualifier.
b) Match on right half of qualifier.
c) Match on entire qualifier.
d) Match only on FSS name.

These options imply a match on FSS name as well.

M is a triplet: file name, qualifier, FP to parameter field.

Note: M specifies a starting point for the search, and the next "match" is the output. If M is null, the search starts at the beginning of the directory.

Outputs: M contains the triplet. IND is the S/F indicator.

BFS PROCEDURES USED INTERNALLY

1) Read key and data of a track.
2) Initialize the primary resident 2314 pack for the BFS.
3) Initialize, terminate, or update the BFS.
4) Change count of open files on a given pack.

TYPICAL FSS FIELD INFORMATION

The following information is shown as the suggested or typical contents of an FSS field of a file index entry.

1) User-specified password.
2) User-specified access information.

3) Pack ID and FP where file is stored, or an FP to a sub-directory containing more information about a file.
4) File size and last-accessed date.
5) User-specified field (similar to the FSS field itself).

IV. A PARTICULAR FILE SUBSYSTEM (FSS1) FOR THE ISPL MACHINE

FSS1: CHARACTERISTICS AND LIMITATIONS

FSS1 (implemented in ISPL) interfaces with users via ISPL calls, assignment statements, and other language constructs. It interfaces with the ISPL Operating System (OS) and one OS component, the Basic File System (BFS), through the ISPL call statement.

Described here is the initial version of FSS1. The interfaces will presumably change in the transition to the ISPL machine. However, the concept of file subsystems and the bulk of this implementation should remain intact.

FSS1 provides random and sequential access, and scatter I/O of variable-length data.

STRUCTURE OF AN FSS1 FILE

A file known to FSS1 has the following structure. A file is composed of variable-length *logical records* (LRs). The LRs are the units for storing and retrieving file data. An LR is composed of variable-length *segments*. Segments of LRs are subunits identifiable for the purpose of specifying parts of an LR in disjointed primary storage locations for scatter read/write. The user may read or write a single segment only if the LR consists of a single segment. That is, the user specifies an LR (not a segment) for an I/O transaction.

IDENTIFYING DATA TO FSS1

A variable-length LR is a collection of data identified by the user to FSS1 for reading or writing. Specifically, with regard to variable length, a given LR may be read, changed in length, and rewritten.

LRs are identified to FSS1 in one of two ways, i.e., by *fixed identifier* (FI) or *file pointer* (FP). When the LR is first written, FSS1 generates an FI once for each LR. FSS1 may output a different FP to the user each time the LR is written.

Fixed Identifier

FSS1 generates an FI (name), unique within the file for each new LR. The user can store and retrieve LRs by name, i.e., FI. Since the FI for an LR does not change, the user can form intra-file (inter-LR) hierarchies where the name of one LR is kept as part of another LR. When using FIs, FSS1 automatically maintains a correspondence between the FI and the current secondary storage location (FP) for each LR of the file.

File Pointer

FSS1 can output an FP each time an LR is written. (The BFS can arithmetically convert the FP to Pack ID and physical secondary storage address.) Because the FP value for a given LR can change each time the LR is stored, it is impractical to use it as an intra-file structural connector. However, the FP is more efficient than the FI.

ACCESS TECHNIQUES

LRs may be accessed *randomly*, *sequentially*, or *indexed sequentially*. For example, an LR is retrieved randomly by specifying either FI or FP; sequentially by requesting the next LR; and indexed sequentially by requesting the *nth* LR. The access technique is specified for individual operations only, as opposed to when the file is "opened."

SCATTER READ/WRITE

An LR may be dispersed in primary storage on input and, likewise, collected from primary storage for output.

The user's invoking process supplies a table of pointers (and lengths for output). Scatter I/O is implied with each read/write operation.

The scattered groups of data are called LR segments. FSS1 records the LR as a contiguous unit on secondary storage; it also appends a header of segment lengths to facilitate the scatter reading operation.

DECLARATIONS FOR PROCESSING LRs

When a file is created, the user must declare the bookkeeping arrangement. The user can request that:

1) FSS1 do bookkeeping: for each write operation only the FI is *normally* output to the user.
2) The user do bookkeeping: for each write both FI and FP are *normally* output.

In either case, FSS1 automatically maintains an ordered list of FI *versus* FP. Regardless of bookkeeping technique, the user may request either FI, FP, or both as output from any given write operation. Thus, the user can save the FI or FP of a specific LR of interest to him in a sequential file, and later treat the file as random for initial positioning. If the FP is supplied on input for a read operation, FSS1 will not reference the bookkeeping table (FI *versus* FP).

FORMATS OF USER FILE OPERATIONS

Users have seven language constructs for file operations (see p. 9). These constructs compile into either direct calls on FSSs, or system calls that invoke the FSSs. Shown below are the user's constructs and the subsystem calls. The parameters unique to FSS1 are described and/or formatted.

1) *Construct*: CREATE FILE (P→A, P→B, C, P→D, P→E, IND)
Compilation: CALL label (P→A, P→B, C, P→D, P→E, IND)

C is a character string (fixed or varying) containing "FSS1." D is a bit string specifying bookkeeping arrangement. E is not used.

2) *Construct*: DESTROY FILE (P→A, P→B, P→D, P→E, IND)
Compilation: CALL label (P→A, P→B, P→D, P→E, IND)

D and E are not used.

3) *Construct*: CONNECT FILE (P→A, P→B, C, P→D, P→E, IND)
Compilation: CALL label (P→A, P→B, C, P→D, P→E, IND)

D is a one-byte integer used as a code for multiplexing over the connection. E is not initially used, but its first byte is reserved for specifying the FSS1-end of the connection for multiplexing.

4) *Construct*: DISCONNECT FILE (P→A, P→B, P→D, P→E, IND)
Compilation: CALL label (P→A, P→B, P→D, P→E, IND)

D and E have the same meanings as in 3), above.

5) *Construct*: FILE (P→A, P→B, P→D, IND) = Q
Compilation: CALL label (P→A, P→B, P→D, IND)

D is diagrammed below. If FI is null, a new LR is assumed. All data shown are inputs except the FP. FP is initially output for each write.

Contents	Length/Type
FI	four-byte integer
FP	pointer
port multiplex code	one-byte integer
unused	three-byte integer
relative LR number	two-byte integer (used only with 6), below)
count of pairs	two-byte integer

	Contents	Length/Type
pair 1	segment addr.	pointer
	segment length	four-byte integer
	.	
	.	
	.	
pair n	segment addr.	pointer
	segment length	four-byte integer

6) *Construct:* Q = FILE (P→A, P→B, P→D, IND)
Compilation: CALL label (P→A, P→B, P→D, IND)

D is the same list shown above. The relative record number is a two-byte signed integer specifying an LR sequentially offset from the current one, i.e., the one named by FI.

If the number of segments in the list is less than the actual number of LR segments, the last segment location specified in the list contiguously receives the remaining LR segments. For example, a scattered LR can be collected by writing it and re-reading it into a single segment location.

7) *Construct:* COMMUNICATE (P→A, P→B, P→D, P→E, IND)
Compilation: CALL label (P→A, P→B, P→D, P→E, IND)

E is not used. D contains a one-byte integer specifying an option. The only currently defined option is to purge an LR. D also contains either the FI or the FP for that LR.

V. DETAILED ORGANIZATION OF THE BASIC FILE SYSTEM (BFS)

THE BFS MASTER FILE

The BFS owns and maintains a BFS master file that describes 1) available and assigned track space for the entire system; 2) information about current FSSs; and 3) a small amount of data about each virtual or real file operating under the subsystems. This section describes the various types of records comprising the BFS master file.

TRACK ACQUISITION FOR THE BFS MASTER FILE

The BFS gets a single track, as needed, and returns tracks as they become unused. All BFS master-file tracks are on resident packs, except for a single track at a fixed location on each pack that describes the track allocation on that pack and gives the pack's identification.

THE PACK IDENTITY AND TRACK MATRIX RECORDS

Each pack contains a pack identity record and a track matrix record. These two physical records occur on the same track at a standard, fixed location for each pack (see Fig. 1).

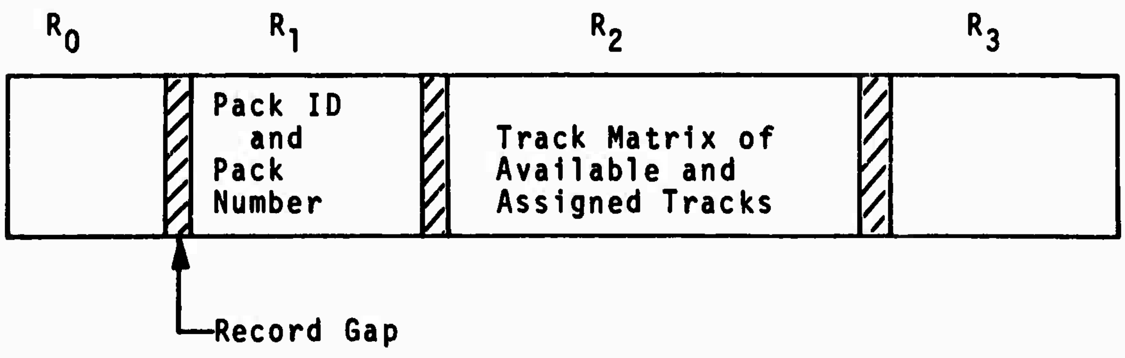

Fig. 1--Pack Identification and Track Matrix Records

The pack ID for a new pack is assigned through a utility program (as yet unspecified). The BFS assigns the pack number.

The track matrix indicates the availability or assignment of each track on the pack. For each assigned track, the FSS to which it is assigned is indicated (see Fig. 2). These matrices are used to allocate tracks to FSSs and to the BFS for the master file.

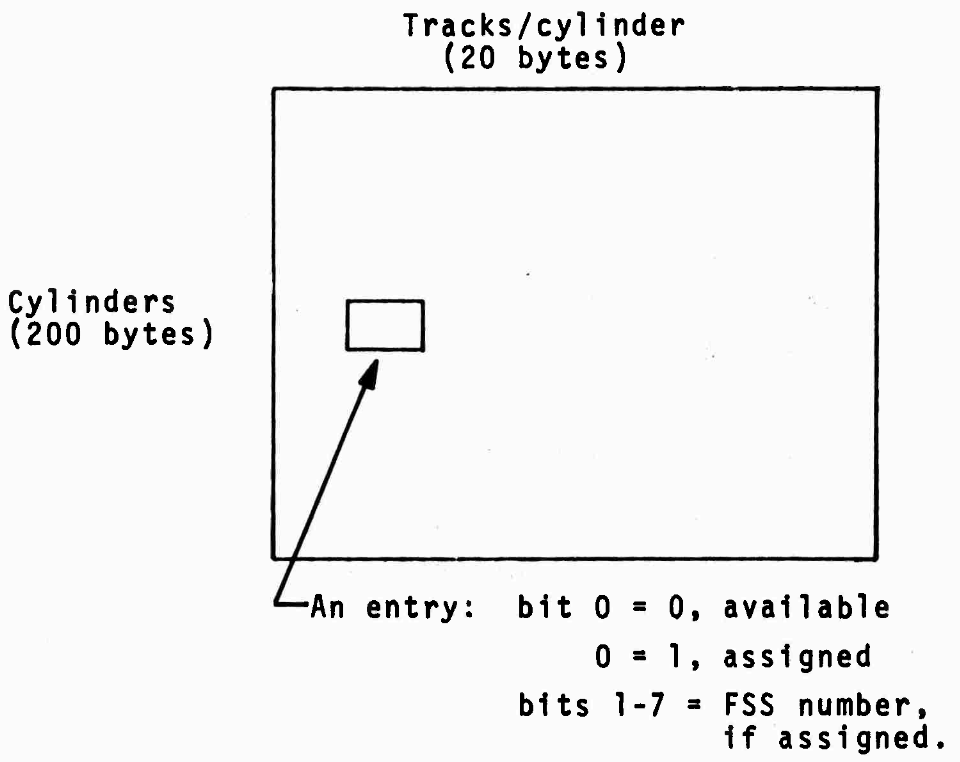
Tracks/cylinder
(20 bytes)
Cylinders
(200 bytes)
An entry: bit 0 = 0, available
0 = 1, assigned
bits 1-7 = FSS number, if assigned.

Fig. 2--Track Matrix

THE MASTER DIRECTORY RECORD

The BFS has a master directory record at a fixed location on a resident pack. This record contains file pointers (FPs) to the current locations of all other BFS records or record types. Excluding the pack ID and track matrix (space assignment), this is the only record of the BFS master file at a fixed location. This record is physical record three (R3) (excluding R0) of the track, which contains the pack ID and track matrix on one of the resident packs. Figure 3 details the contents of this record.

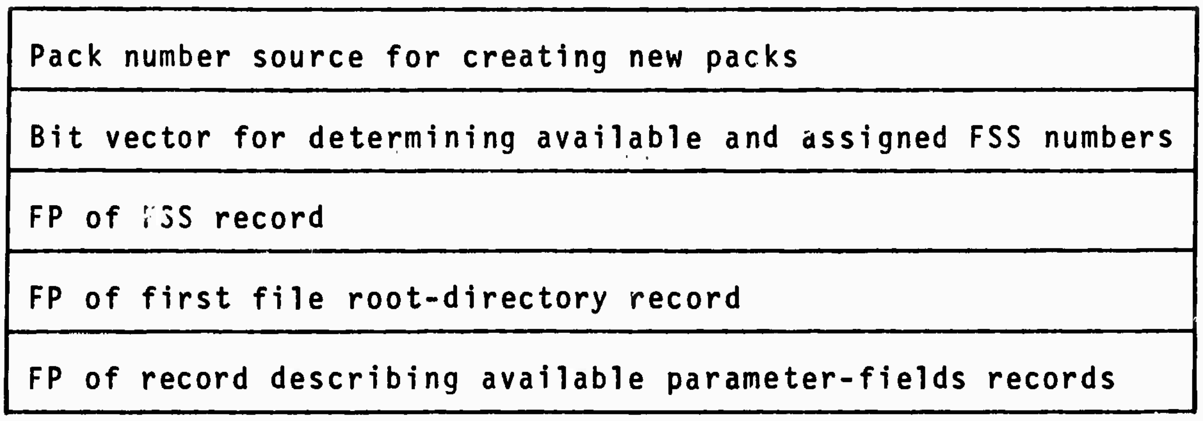
Pack number source for creating new packs
Bit vector for determining available and assigned FSS numbers
FP of FSS record
FP of first file root-directory record
FP of record describing available parameter-fields records

Fig. 3--Master Directory Record

THE BFS RECORD OF FSSs

The BFS maintains a record on a resident pack of current FSSs. It is formatted as one physical R1 of maximum length; Fig. 4 shows its contents.

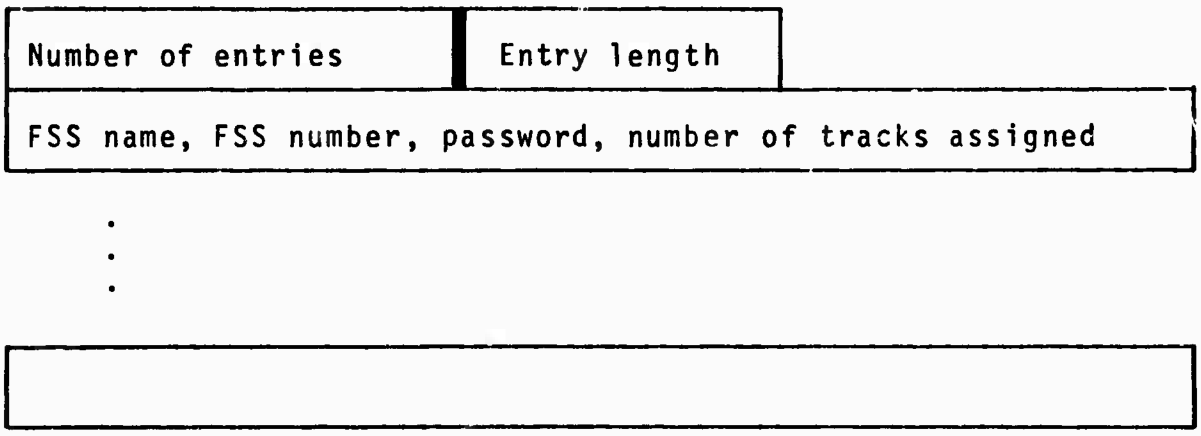

Fig. 4--File Subsystems Record

THE BFS FILE ROOT-DIRECTORY: FORMAT AND CONTENTS

The root-directory record(s) contains all the information of a file index entry except the variable-length parameter field. Its format is one maximum-length R1 per track (see Fig. 5).

THE BFS RECORDS OF FILE INDEX PARAMETER FIELDS

Tracks are obtained and released as needed to house the variable-length parameter fields of file index entries. The tracks are formatted with a fixed number of equal-length physical records/track. Each record can contain 32 bytes of parameter-field description. For file index entries with parameter-field entries longer than 32 bytes, an additional read is required for each 32-byte segment (see Fig. 6).

THE BFS AVAILABLE PARAMETER-FIELDS DESCRIPTOR

The available parameter-fields descriptor is formatted as a maximum-length R1 on a resident pack. It describes the available and assigned parameter-field-segment records of tracks assigned for that use (see Fig. 7).

The record subfield of the FP is zero. It is modified by a position within the bit vector. The bit vector states availability of each physical parameter-field record on the track addressed by the FP.

DATA MAINTAINED IN PRIMARY STORAGE BY THE BFS

The BFS dynamically maintains a table of current pack number *versus* drives. An entry is made for a given drive each time the drive is enabled, and deleted each time it is disabled. The table is a vector of pack numbers indexed on drive number. The BFS also keeps a copy of the master directory record in primary storage.

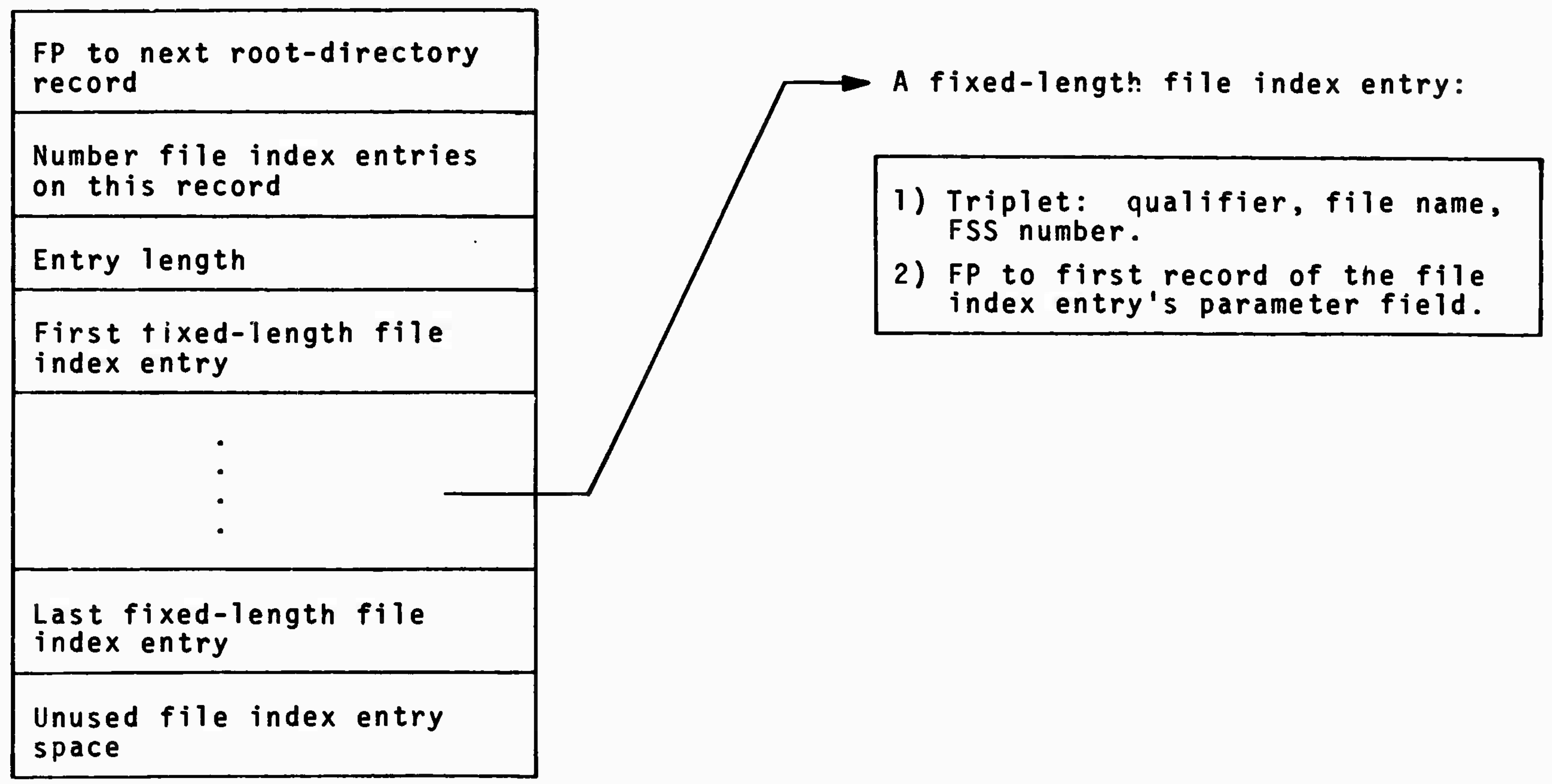
FP to next root-directory record
Number file index entries on this record
Entry length
First fixed-length file index entry
.
.
.
.
Last fixed-length file index entry
Unused file index entry space
A fixed-length file index entry:
1) Triplet: qualifier, file name, FSS number.
2) FP to first record of the file index entry's parameter field.

Fig. 5--File Root-Directory

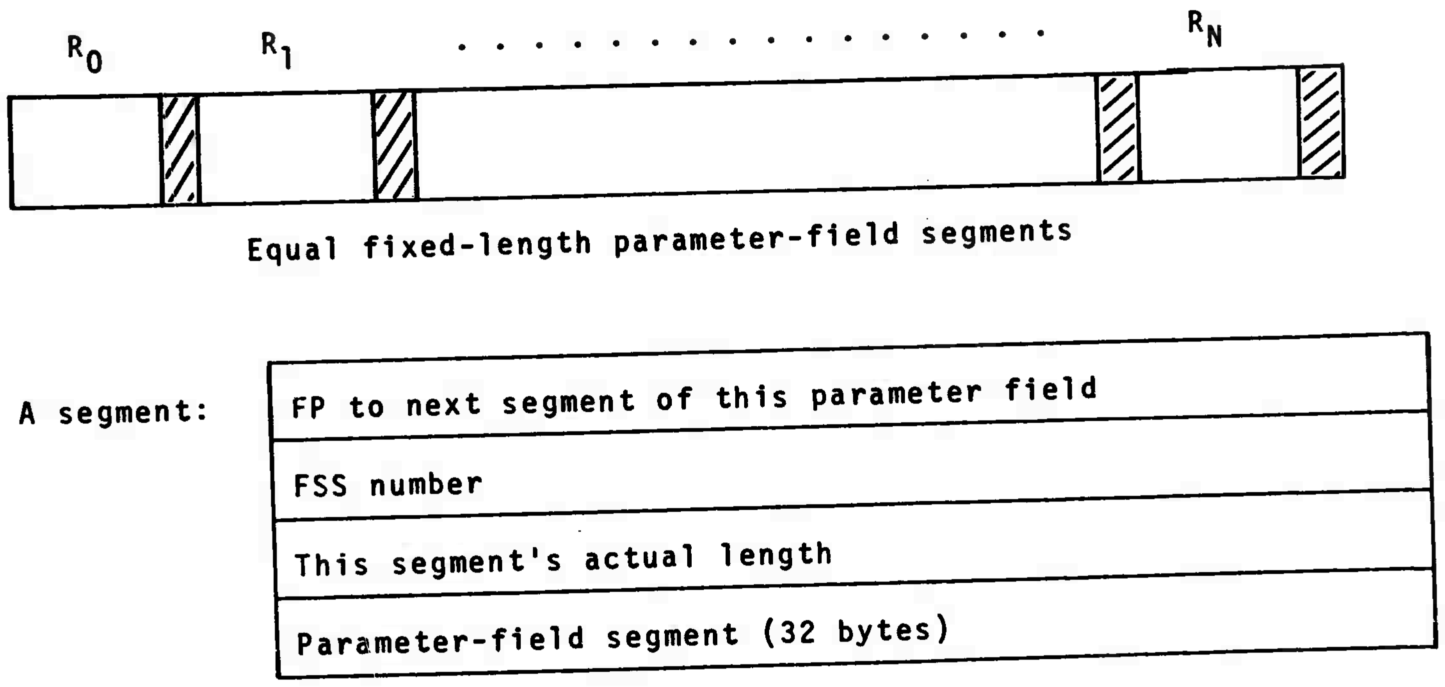
R0
R1
RN
Equal fixed-length parameter-field segments
A segment:
FP to next segment of this parameter field
FSS number
This segment's actual length
Parameter-field segment (32 bytes)

Fig. 6--File Index Parameter Fields

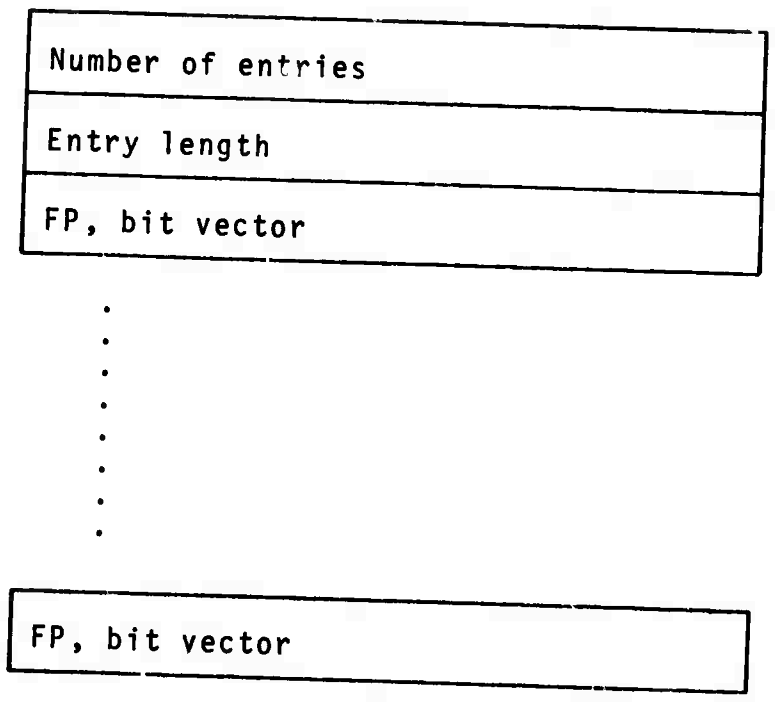

Fig. 7--Available Parameter-Fields Descriptor

VI. DETAILED ORGANIZATION OF THE FSS1 SUBSYSTEM

THE FSS1 MASTER FILE AND USER FILES

The FSS1 subsystem owns and maintains a master file that describes available and assigned track space for all files operating under FSS1. The FSS1 master file has exactly the same overall organization as user files that operate under FSS1; that is, the master file is accessed by FSS1 using the same constructs users employ to access their files.

TRACK FORMAT FOR FSS1-TYPE FILES

All tracks assigned by the BFS to the FSS1 subsystem are formatted alike and have similar organizations. Figure 8 shows the format and organization.

THE FORMAT OF FSS1 SUBSYSTEM LOGICAL RECORDS

Figure 9 shows the format of a logical record (LR) under the FSS1 subsystem.

FILE CODES

Each file created under FSS1 is assigned a file code that is stored in the parameter field of the file index entry in the BFS master file. The source of available and assigned file codes is kept as a 128-bit vector in the parameter field of the file index entry of the FSS1 master file.

THE PARAMETER FIELD OF THE BFS'S FILE INDEX ENTRY FOR THE FSS1 MASTER FILE

Each file has a file index entry in the BFS master file. Figure 10 shows the parameter-field part of that entry for the FSS1 master file.

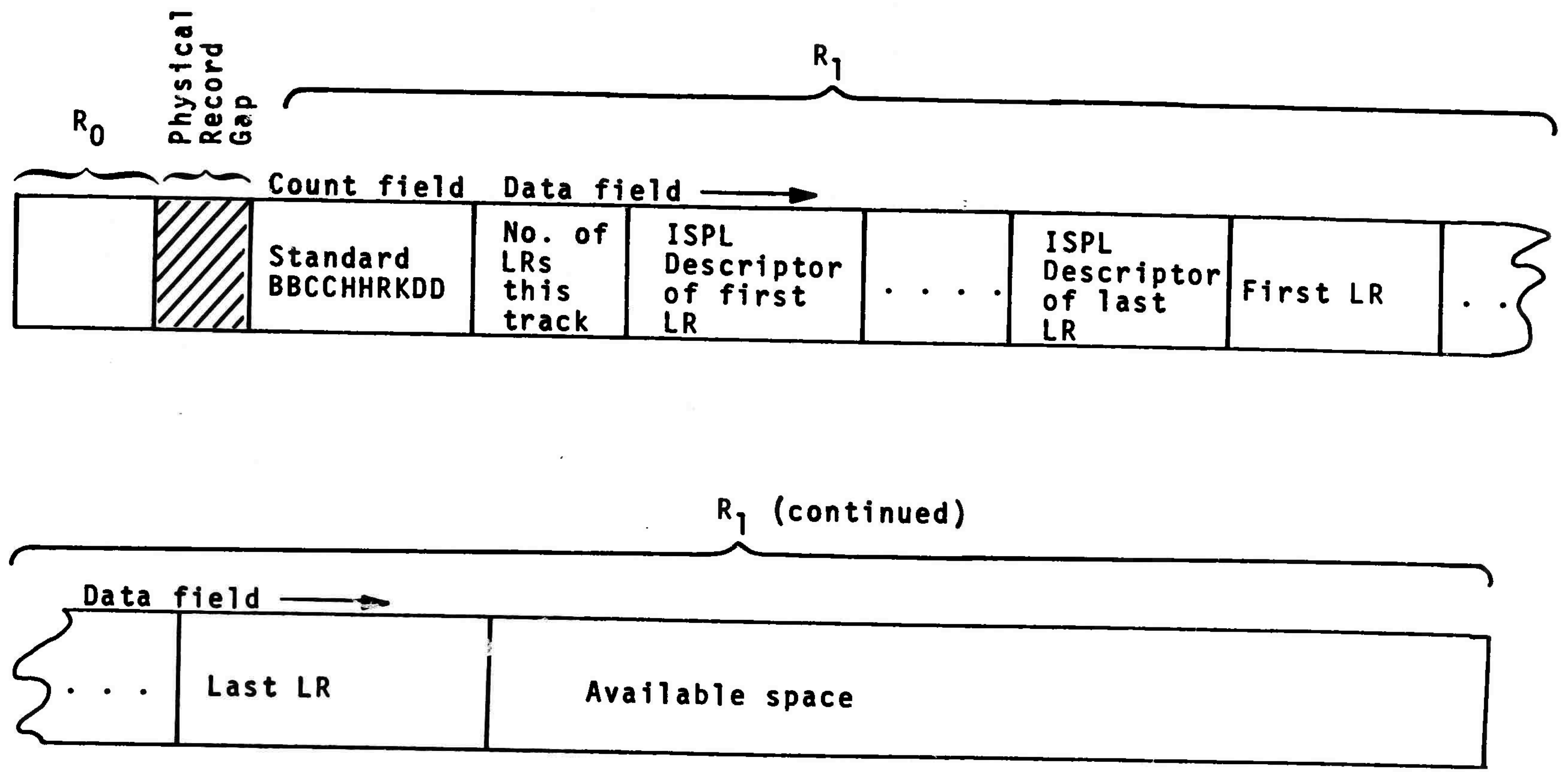

Fig. 8--Track Format and Organization

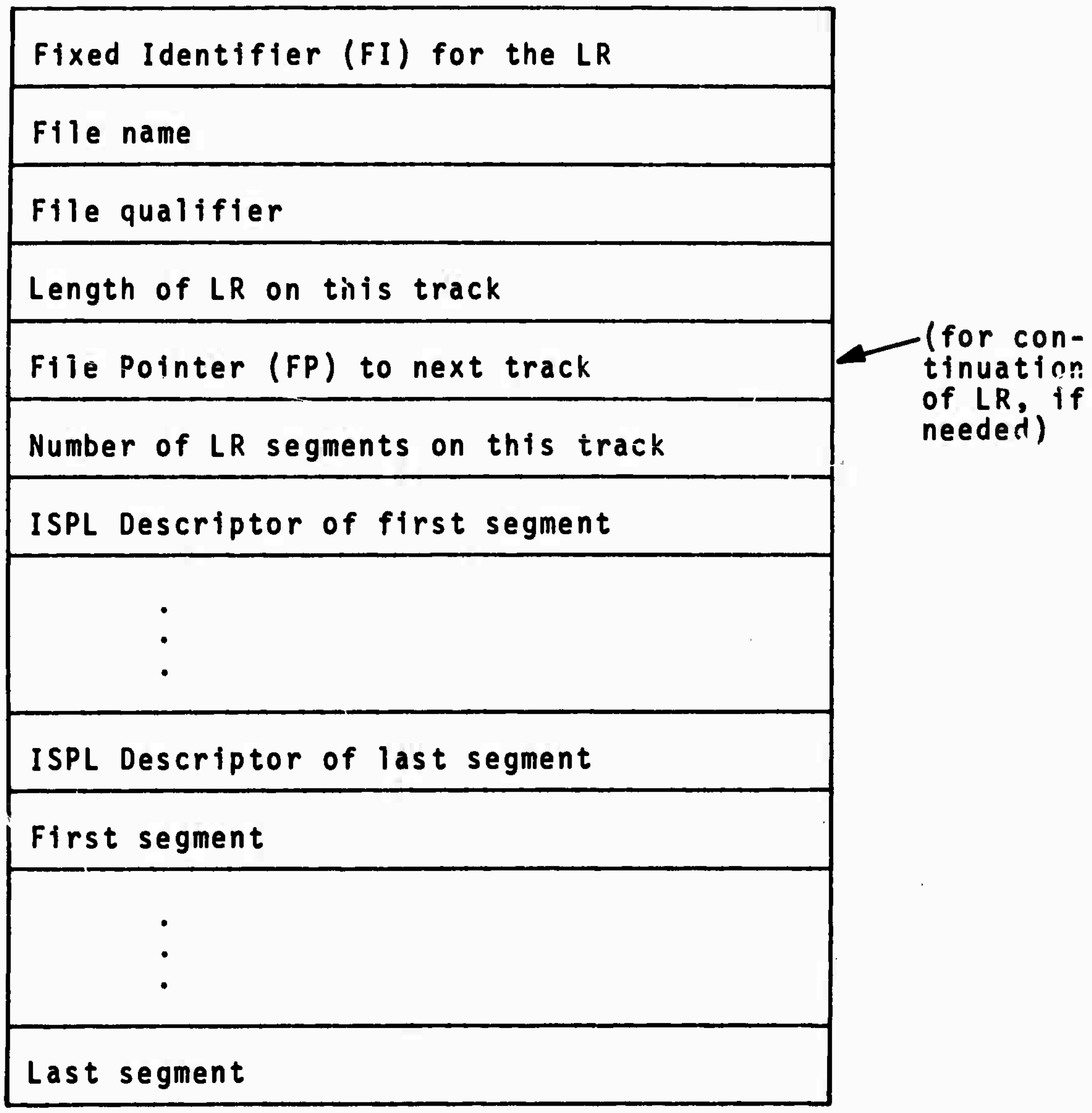
Fixed Identifier (FI) for the LR
File name
File qualifier
Length of LR on this track
File Pointer (FP) to next track
(for continuation of LR, if needed)
Number of LR segments on this track
ISPL Descriptor of first segment
ISPL Descriptor of last segment
First segment
Last segment

Fig. 9--Logical Record Format

FI source for LRs in the master file
The FI, FP for the LR containing the Bookkeeping and Partially Available Space for the master file
The FI, FP for the LR containing the track matrix for all FSS1 subsystem files
A bit vector denoting assigned and available file codes

Fig. 10--FSS1 Master File's Parameter Field

PARAMETER FIELD OF FILE INDEX ENTRY OF ALL FSS1 USER FILES

Each file has a file index entry in the BFS master file. Figure 11 shows the parameter-field part of that entry for user files under FSS1.

FI source for LRs in the file
FI, FP for the LR containing the Book-keeping and Partially Available Space information
The file code

Fig. 11--User's Files Parameter Field

TRACK ACQUISITION FOR FILES OPERATING UNDER FSS1: THE FSS1 MASTER FILE'S SPACE ALLOCATION LR

FSS1 gets a block of contiguous tracks, as needed, from the BFS for further suballocation to the FSS1 master file and its user's files. An LR in the FSS1 master file describes this space; its format is shown in Fig. 12.

FP to first track	Number of tracks	Byte vector

The byte vector contains a byte for each track. Its format is:

bit 0 = 0, available
bit 0 = 1, assigned
bits 1-7 = file code to which track assigned

These fields are repeated for each contiguous block of tracks.

Fig. 12--FSS1 Master File Space Allocation LR

SPACE ALLOCATION BY FSS1

FSS1 assigns space, in units of a track, to individual files as needed. When a track is assigned, it is so marked in the byte vector of the Space Allocation LR in the FSS1 master file, and the file's code is inserted in the low-order bits of the byte.

As tracks become available (due to rewriting or purging of LRs), they are returned to FSS1 and so marked in the appropriate byte of the Space Allocation LR in the FSS1 master file.

THE BOOKKEEPING AND SPACE ALLOCATION LR AND ITS USAGE

There is an LR for each FSS1 file (including the FSS1 master file). This LR contains 1) the bookkeeping correspondence of FI *versus* FP for each LR of the file, and 2) a partially available space count for each track assigned to the file. When a file is created, FSS1 constructs this LR with one FI *versus* FP entry (for the record itself), and one partially available space entry for the track on which it is written. The BFS maintains the FP for this LR in the parameter field of the file index entry. Figure 13 shows the format of this LR.

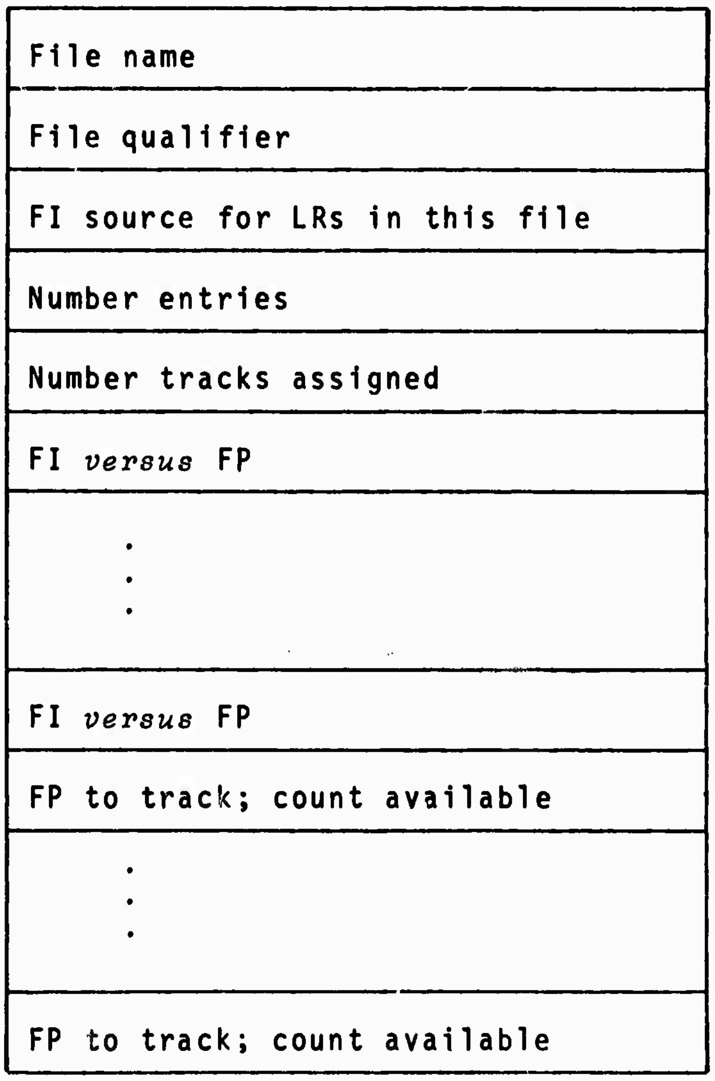
File name
File qualifier
FI source for LRs in this file
Number entries
Number tracks assigned
FI versus FP
⋮
FI versus FP
FP to track; count available
⋮
FP to track; count available

Fig. 13--Bookkeeping and Partially Available Space LR

This LR is updated each time an LR is purged or rewritten in a new physical location; it is also updated each time a new LR is written into the file.

The first half of this LR is used to obtain FP, given FI. The second half is used to:

1) Purge LR--all freed space is added to the counts for the appropriate tracks.
2) Rewrite LR in a new location--if LR is same size, rewrite in same location; if not, free space from old LR and do 3), below.
3) Write a new LR--a) If more than one track is needed, get it from FSS1 and add an entry with FP, count = 0. Repeat the above until less than one track is needed. b) Search the FP *versus* free-count table for an amount ≥ the residual. If not successful, repeat part a), above. c) Deduct the amount needed for the residual. d) Write the LR.

In cases 1) and 2), if the count reaches a complete track, delete the FP *versus* free-count entry and return the track to FSS1's available space.

OPEN FILES TABLE

In a primary storage ISPL record, FSS1 maintains a table with an entry for each open file. An entry contains:

1) File name.
2) File qualifier.
3) Process name or port name and multiplex code.
4) Current FI, FP record read.
5) Current FI, FP record written.
6) FP to the parameter field in the directory.
7) The parameter-field contents.

FSS1 PRIMARY STORAGE USAGE

1) The Open Files table is maintained in primary storage.
2) The Bookkeeping and Partially Available Space LR for the most recent transaction is in primary storage. It is rewritten *at least* after each CREATE, DESTROY, and CLOSE operation. Since the Bookkeeping/Partially Available Space LR last used is kept in primary, a user who writes or reads in a burst may not need to repeatedly bring this record into core.

OPERATIONS REQUIRED FOR FSS1 FUNCTIONS

1) CREATE a file
   a) Create a BFS file index entry.
   b) Write a Bookkeeping/Partially Available Space LR in the file.
   c) Write the FSS1 Space Allocation LR.
2) DESTROY a file
   a) Mark all tracks assigned to this file available in the FSS1 Space Allocation LR; rewrite this LR.
   b) Destroy the BFS file index entry.
3) OPEN a file
   a) Retrieve the BFS file index entry.
   b) Add an entry to Open Files table.
4) CLOSE a file
   a) Update the BFS file index entry.
   b) Delete the entry in Open Files table.
   c) Write the FSS1 Space Allocation LR.
5) SEND
   a) Bring Bookkeeping/Partially Available Space LR into primary if not there.
   b) If a new LR, go to e), below.

c) Read in old LR.

d) If same size, insert new LR, write, and exit; if not, pack physical record rewrite, and indicate space freed.

e) Allocate space as described on p. 34.

f) Write the record.

g) Add new FI *versus* FP entry, as appropriate.

6) RECEIVE

a) If given FI, FP, go to d), below; if not, continue.

b) If it is not already there, bring Bookkeeping/ Partially Available Space LR into primary.

c) Retrieve FP.

d) Read the record.

7) PURGE LR

a) If it is not already there, bring Bookkeeping/ Partially Available Space LR into primary.

b) Read the record.

c) Update count of free space.

d) If whole track is free, return it to FSS1, delete FP *versus* free-count entry and exit.

e) If track is not free, compress and rewrite LR.

REFERENCES

1. Balzer, R. M., *The ISPL Machine: Principles of Operation*, The Rand Corporation, R-562-ARPA, August 1971.

2. -----, *The ISPL Language Specifications*, The Rand Corporation, R-563-ARPA, August 1971.

3. Ellis, T. O., J. F. Heafner, and W. L. Sibley, *The GRAIL Logical Input/Output Processes*, The Rand Corporation, RM-6257-ARPA, May 1970.