NCR CONTROL DATA CORPORATION

ADVANCED SYSTEMS LABORATORY

CHAPTER 05

DATA MANAGEMENT

Doc. No. ASL00282

Rev. 04

Copy No. 87

TABLE OF CONTENTS

1.0 INTRODUCTION

The IPL Data Management system consists of operating system procedures which have the general responsibility for managing files, peripheral devices and removable storage volumes and for moving data between the peripheral devices and central memory. The Data Management system consists of five major functional areas: Volume Management, File Management, Record Management, Block Management and Device Drivers. Data Base Control Procedures provide a higher level capability than the Data Management system and are described in separate specifications.

Volume Management procedures operate in close cooperation with Job Management procedures to control the scheduling of peripheral devices and attachment/detachment of removable storage volumes to/from peripheral devices.

File Management procedures have the general responsibility for all operations necessary to prepare a file for processing and for all operations necessary to clean up after file processing is complete. These operations fall into the general categories of file definition and creation, file catalog management and execution-time file management.

Record Management procedures have the general responsibility for providing storage and retrieval of records in a variety of file organizations. They are the primary execution-time interface between the user and a file.

Block Management procedures have the general responsibility for coordinating the movement of data blocks between the user and a file and for routing external signals back to the user. Block Management procedures also control the logging of abnormal conditions detected by the peripheral system.

Device Drivers have the general responsibility for managing the communication with the hardware/software components of the peripheral configuration, for first-level error recovery and for capturing all signals coming in from the peripheral configuration. Device Drivers are described in separate specifications.

1.1 DESIGN OBJECTIVES

A number of assumptions and objectives influence both the features that are present in the Data Management system and the internal structure that supports the features. Some of the more important of these are discussed in the following paragraphs:

- The Data Management system must meet the requirements imposed by ANSI standards.
- The Data Management system must support the requirements imposed by members of the IPL product set unless a specific requirement is unique to only one product set member. In that case, some judgement may be applied to determine whether the requirement should be satisfied by the Data Management system directly or if the product set member should/could satisfy the requirement using some combination of existing Data Management facilities.
- The Data Management system must provide support for sharing of files among multiple users, each of whom may be updating the file.
- The Data Management system must provide the capability for an installation to interpose a security subsystem between users and the Data Management System.
- The Data Management system must be capable of effectively using extremely large central memories without design change.
- The Data Management system must be capable of operating on a system which has a central memory of 256K bytes.
- The Data Management system must be capable of effectively supporting very large mass storage files (exceeding 4x10**9 bytes).
- The Data Management system must efficiently support a large installation which operates in a multiprogramming and multiprocessing mode with on-line storage on the order of one billion bytes, on-line files totalling on the order of ten thousand, and some form of off-line or removable storage totalling several times the capacity of the on-line storage.
- The Data Management system must permit tape and disk

storage volumes to be easily interchanged among IPL installations.

o The Data Management system must support terminals in a manner consistent with support of other hardware types. To the maximum practical extent, IPL programs should be able to deal with terminals and unit record devices as sequentially organized files.

1.2 TERMINOLOGY

Record - A record is the unit of addressability of a file through a Record Management procedure. A record consists of a fixed or variable length byte string whose content is generally known only to the user. The only interpretation of record content by a Record Management procedure is on fields which have been defined by the user to contain keys to be used for subsequent addressing. Five record types are supported, four of which conform to ANSI tape interchange standards. The fifth is the default system format.

Physical Record - A physical record is the smallest unit of data transfer between a device and memory for which a device status is available. Cards, print lines, tape records and disk sectors are examples of physical records.

Block - A block is the unit of addressability of a file through a Block Management procedure and is also the unit of transfer between central memory and a file. The maximum block size is fixed for a file at the time the file is first defined. Block Management procedures are unaware of the content of a block and treat it as an unformatted byte string to be mapped onto a peripheral recording device. The mapping is done in one of three basic ways, depending on the characteristics of the recording device.

1. If a file is on magnetic tape, then each block is recorded as one physical tape record. Blocks within a file may vary in length up to the user-specified maximum.
2. If a file is on a mass storage device, each block begins at a physical record (sector) boundary and continues through as many contiguous sectors as necessary. Blocks within a file are mapped as fixed-length units of storage which may contain variable amounts of data.
3. If a file is on a unit record device, each block begins on a physical record (card, print line, etc.) boundary and continues through as many physical records as needed. If blocks contain multiple records, the filling and emptying of blocks from/to the unit record device is accomplished by device drivers.

File - Depending on the users level of interface to the Data Management system, a file may be viewed in the following ways:

1. File Management Level - A file consists of a peripheral device or a region of storage on a volume.
2. Record Management Level - A file consists of a set of records addressable by key, by ordinal, or by file address.
3. Block Management Level - A file consists of a set of blocks addressable by block numbers.
4. Segment Level - A file consists of a segment of virtual memory addressable by segment number and byte offset.

File Control Block - A File Control Block is a symbolically addressable table in a System LNS segment that contains a description of the logical properties of a file. A File Control Block is the primary means through which parameters are passed from the user to the File Management procedures.

Permanent File - A permanent file is a mass storage file for which a description of its logical and physical characteristics has been recorded in a catalog. Permanent files survive beyond the life of the job in which they are created.

Temporary File - A temporary file is one for which a description of its logical and physical characteristics exists only in the File Control Block and internal system tables. Temporary files exist no longer than the life of the job in which they are created. A temporary mass storage file may be converted to a permanent file.

File Address - Each time a record is stored in or retrieved from a mass storage file, the Record Management procedures return a file address which uniquely identifies the record. Users may save this file address and use it later to retrieve the record. The file address of a record remains constant until the record is deleted or the file is reorganized.

Device - A device is an addressable unit in the peripheral configuration that can be acquired for use by a job. Examples of devices are tape drives, disk storage drives, card readers, punches and printers.

Device Control Block - A Device Control Block is a symbolically addressable table in a system LNS segment that contains a description of a device.

Volume - A volume is a unit of external storage such as a fixed-head disk, a drum, a disk pack or a reel of magnetic tape. Each volume of a given type is identified by a

six-character alphanumeric serial number which is recorded (magnetically) in a volume label and is also printed externally on the volume. Magnetic tapes which are defined to have non-standard or no labels are accomodated as exceptions.

Volume Set - A volume set is a logically related set of volumes which contains a file or a set of files. A single file catalog exists for each mass storage volume set. Files can not span volume sets. A volume cannot be a member of more than one volume set. A mass storage volume set cannot be partially on-line if files are being processed on it.

Volume Set Control Block - A Volume Set Control Block is a symbolically addressable table in a System LNS segment that contains a description of a volume set.

System Volume Set - The System Volume Set is a mass storage volume set that is permanently on-line and is implicitly attached to every job. It provides the default residency for all permanent and temporary mass storage files. Its size is selectable by an installation and must consist of at least one volume.

Labels - Two types of labels are processed by the Data Management system; volume labels and file labels. Standard volume labels for magnetic tape conform to ANSI interchange standards. Standard volume labels for mass storage carry additional information required to describe the volume. Standard file labels for magnetic tape conform to ANSI interchange standards. Labels for mass storage files are recorded in volume set catalogs and are not compatible with any known system. The Data Management system also supports tapes with non-standard or no labels.

1.3 INCOMPLETE FUNCTIONAL AREAS

A number of functional areas are either incompletely specified or are relatively unstable at this date. In either case, the information relating to these areas in this issue of the GDS can be expected to change, either because information is replacing an absence of information or because stable information is replacing unstable information. GDS changes which perform the reverse are not planned. Relevant functional areas are described in the following paragraphs.

Terminal Processing - A consistent interface between programs and terminals, the general capabilities of various software components (message control systems, front-end processors, communication controllers, device drivers, block and record management procedures, etc.) and the hardware components of the peripheral configuration are not yet specified.

Hardware-Dependent Interfaces - Most peripheral hardware characteristics visible through the interfaces discussed in this issue of the GDS are based on general hardware characteristics rather than on IPL hardware specifications. As conroller and device characteristics for IPL become specified, these areas of the GDS can be expected to change.

Management of Output Streams - The command language chapter of the GDS discusses a stream capability which permits a many-to-one and one-to-many relationship between streams and files. Because of its suitability for logging and handling the many system-supplied job files, this capability is intended to be eventually incorporated into the Data Management System. We haven't had time to do it yet.

Tape Label Processing - This area is not yet well defined. Some resolution of Cobol requirements and a more complete specification of tape label processing is expected to occur during the next few months.

Generation Number Processing - The requirement for this facility is not cleary defined. As it becomes better defined, the GDS will be modified to reflect the new requirements.

File-Program Relationships - The general use of the execute attribute on files, the use of Library file organization, the use and meaning of opening a file for execution, the assigning of execute ring brackets to files and the exact relationship between files and segments are all undergoing

review at the present time. The capabilities described in this GDS are not necessarily expected to change but the method of achieving the capabilities may change.

File Sharing - Topics that do not yet appear to have stabilized include: a concise definition of the relationships among jobs, control points, instances of open and locked blocks or records; proper resolution of deadlocks; and a concise description of valid concurrent file sharing selections.

Device-Volume Sharing - Clarification of the relationships between the Job Management scheduling requests and device-volume-file requests will occur over the next few months.

2.0 BASIC CONCEPTS

The following topics are judged to be sufficiently significant to justify their appearance in this section rather than the section on terminology.

2.1 FILE DEFINITION

The File Control Block provides the primary interface through which a user supplies the definition of a file. A File Control Block, which may be constructed either by direct use of LNS requests or by use of the File Management FM#DEFINE_FILE request, must exist in a System LNS segment before a file can be referenced.

When the File Control Block is first constructed, all fields are set to null values. Values subsequently supplied through LNS requests either establish an initial value or override an existing value. Values supplied through the FM#DEFINE_FILE request do not override existing values. When the file is subsequently created (mass storage) or opened (non-mass storage), default values are supplied for all null fields. Attaching a permanent mass storage file causes the values in the corresponding cataloged fields to override existing File Control Block values (a few exceptions are noted in the description of the FM#ATTACH_FILE request). The File Control Block is locked at the time the final values are supplied (when the file is created, attached, or opened) and cannot then be directly modified by the user.

In summary, the following items are important:

a) a File Control Block must exist before a file can be referenced

b) cataloged values can replace any existing File Control Block values

c) values supplied through LNS requests can replace any existing File Control Block values

d) values supplied through the FM#DEFINE_FILE request can replace only null values

e) no File Control Block fields can be directly modified by the user after the file is created (new mass storage files), attached (existing mass storage files) or opened (non-mass storage files).

The precedence rules are based on the assumption the file definitions will be supplied through the System Command Language by use of LNS requests and through compile-time declarations by use of the FM#DEFINE_FILE request. This permits compile-time declarations to be overriden by the user at execution time, through use of the System Command Language, but assures that the cataloged description of the file will override both of the others.

2.2 FILE SECURITY

File security relies heavily on the validation of a user's identity at the time the user logs into the system. The primary objective is to concentrate on verifying that the current user is, in fact, who he says he is, then rely on that verification to control access to files.

A second aspect of file security relates to the internal structure of IPLOS components and to the use of the IPL memory protection hardware. Given that the owner of a file is entitled to do anything he wishes with the file, the major problem is to prevent others from doing things the owner does not want them to do. By (a) allowing the owner to specify the type of access he will permit others to have (e.g., shared read-only access through a specific File Access Procedure which runs at a privileged subsystem level), then (b) placing the data and related control tables into memory segments which are not directly accessable to the user, then (c) forcing access requests to be made through controlled entry points and verifying that all entry points are followed in the proper order, the specified level of security is achieved without unreasonable performance penalties.

The use of a File Access Procedure, discussed as a separate topic in this section, permits the owner of a permanent mass storaje file to build additional security checking if needed since the Data Management software forces every request against the file to be routed through the File Access Procedure before being processed by the system.

2.3 FILE OWNERSHIP

The owner of a permanent mass storage file is identified at the time a catalog entry is generated for the file and is defined to be the user whose identity resides in the current Job Control Block (the user identifier supplied with the Login request). The owner of a file is entitled to issue all file management requests against the file and can assign or deny to other users the right to attach and open the file. Since the owner of the file is also the creator of the initial File Control Block for the file and, therefore, is the creator of the cataloged descriptor for the file, he can be assured that subsequent users of the file will operate with a copy of the original File Control Block. This permits the original access level, File Access Procedure and other processing options to be repeated on all subsequent accesses.

2.4 ACCESS CONTROL LISTS

The owner of a permanent mass storage file may generate an Access Control List which allows others to access the file and specifies in what manner they may acess it. Each entry in an Access Control List has three components; a user identifier, an account identifier, and the access rights allowed for this combination of user identifier and account identifier. By convention, a null value for either of the first two components is defined to mean "any user" or "any account". This allows the following valid combinations to be defined:

User	Account	Meaning
M	N	user M under account N
M	null	user M under any account identifier
null	N	any user under account N
null	null	any user under any account identifier

The (M,N) combination is, of course,most specific. The (null, null) combination defines a public file if any access rights are enabled or a private file if no access rights are enabled. Allowable access rights are:

Exclusive - No other concurrent users are allowed
Protected - Other concurrent readers are allowed
Unprotected - Other concurrent readers and writers are allowed
Input - Data may be retrieved from the file
Output - The file may be written from its beginning
Extend - Data may be appended to the file
Update - Data may be retrieved/inserted/deleted/modified
Execute - The program in the file may be executed

These rights specify only the type of access which the users identified by this entry are entitled to request when the file is attached or opened. The users are not guaranteed they can actually exercise a right when they eventually ask to do so since another user may have a conflicting access right in effect at that time. For example, a file can not be attached for exclusive use if it is currently attached by another job, even though the user is entitled to exclusive access.

An Access Control List is internally ordered and processed in such a way that individual entries are checked before group entries. Access rights are selected from the first entry which satisfies the user/account combination.

2.5 FILE SHARING

A major design objective of the Data Management system is to provide adequate support for processing of shared files. Traditional support for shared read-only or read-execute files is relatively straightforward since the content of the file is not modified. The major design requirement is simply to permit multiple copies of the necessary run-time control tables to be established.

However, support for shared files which can be modified by concurrent users requires solutions for a totally different set of design requirements. For example, a single shared copy of many of the internal control tables is required, queueing logic to permit proper sequencing of conflicting demands is required, detection of deadlock conditions must be provided, new interpretations of traditional "currency" information are required and, finally, an external interface must be defined to allow users of shared files to specify the kind of shared environment they are willing to accept. All of these solutions must not, of course, impose an excessive overhead on the large majority of users who do not require the service.

Sharing is supported only for permanent mass storage files. At the time a permanent mass storage file is attached to a job, one of three sharing options and one of five usage options are specified. These selections identify the way in which the requesting job wishes to share and use the file. After the file is opened, explicitly selectable locking conventions are defined to permit data accesses to be coordinated.

Sharing and usage options are defined in section 2.4 and in the relevant request descriptions. Appendix B summarizes the valid combinations of concurrent sharing and usage options. Locking conventions are defined in the sections on record management requests and block management requests. An expanded definition of the three sharing options appears below.

Exclusive - Only one job may be attached. Only one concurrent open is permitted. Explicit lock selections are not required and are ignored if issued.

Protected - Multiple jobs may be attached but no <u>other</u> job can modify data in the file. Multiple concurrent opens are permitted, only one of which (in the current job, of course) may allow data modification. Explicit lock selections are not required and are ignored if issued.

Unprotected - Multiple jobs may be attached and any job can modify data in the file. Multiple concurrent opens are permitted and any of them may allow data modification. Explicit lock selections are honored by the system and are mandatory if data is to be modified (records may not be replaced or deleted unless previously locked).

2.6 FILE ACCESS PROCEDURES (FAPS)

File Access Procedures are procedures which may be inserted into the normal flow of control between the user and the data management system. FAPs may execute at either the user level or the subsystem level and typically will be employed to extend the apparent capability of the data management system.

When a file is defined, the owner of the file may select an associated FAP by specifying only its entry point (if at the user level) or an entry point within a specified subsystem that has been registered with IPLOS. Subsequent data management requests for that file will be routed, by standard linking conventions, to the FAP. The FAP has complete freedom to choose whether to pass the request on to the data management system, to process the request itself, or to issue additional requests in place of the original request. Although the standard linkage conventions can be bypassed by a call issued directly from the user to the system, this condition is detected (if the FAP executes at the subsystem level) and the illegal call is rejected.

Major anticipated uses for user-level FAPs include:

a) processing non-IPL data formats on interchange media.
b) generation of logging or audit trail files
c) simulation of data files for checkout purposes

Major anticipated uses for subsystem-level FAPs include:

a) provision of the IPL Data Base Management facilities
b) installation-supplied security procedures

The data management system permits a single FAP to be definea for a given file. If multiple non-standard functions are required to be applied to the file (logging and code conversion, for example) then the multiple functions must be provided by a single FAP.

NOTE: The user should be aware that a FAP, since it receives control between the user and the system, may choose to modify the user's initial description of a file. The File Control Block will then contain whatever the FAP wishes it to contain.

2.7 ACCESS LEVELS

The Data Management system provides three levels through which a file may be accessed. Record-level access means that the unit of transfer between the user and the Data Management system is a record. Buffering, imbedded control information and most device dependencies are managed by the system. Block-level access means that the unit of transfer between the user and the data management system is a block. The user assumes responsibility for buffering, interpretation of imbedded control information and is increasingly aware of device dependencies. Segment-level access is available only for mass storage files. Here, the Data Management system maps the file to a virtual memory segment and the user directly references the data with machine instructions. The movement of data between the virtual memory segment and the external file is managed through the standard system paging logic. When a user selects record-level access, the system record management procedures will, in turn, use either block-level or segment-level access.

When a file is first defined, the owner may specify which of the three access levels is to be used to initially write the file (FAPs, in effect, provide a fourth, higher, access level) and may establish the conditions under which the file may subsequently be accessed at another level. The later selection of an access level, different from the one through which the file was originally written, is accomplished by making the appropriate selection in a File Control Block before attaching the file. The control on the legality of these selections is accomplished through the use of four ring numbers which were placed in the original File Control Block. The first specifies the highest ring number from which record-level requests may be issued. The second specifies the highest ring number from which block-level requests may be issued. The third and fourth specify the read/write (or execute) bracket to be placed on the virtual memory segment if segment-level access is selected. If no ring values are specified when the file is initially defined, the system supplies default values to enforce the original access level.

2.8 FILE ORGANIZATION

The organization of a file defines its logical structure, is established when the file is first defined, and cannot subsequently be changed. With the exception of Library organization, a file organization is managed by standard Record-Level access procedures which are concerned with maintaining proper record relationships. At lower levels (block or segment level) a file looks like a string of bits and file oganization has little, if any, meaning. Four file organizations are currently defined: Sequential, Relative, Indexed and Library.

Sequential file organization is supported on all recording media and is one in which predecessor-successor record relationships are determined by the order in which records are placed into the file. Each record, except the first, has a unique predecessor. Each record, except the last, has a unique successor. A sequentially organized file on mass storage may be updated in place (records may be replaced) if the original record length is not changed. If a sequentially organized mass storage file contains Y-format records, records may also be logically deleted even though the record remains physically present and continues to occupy space.

Relative file organization is supported only on mass storage devices and consists of fixed-length entries (addressed by the ordinals 1, 2, ..., N) which contain Y-format records. When the file is created, the entry size is selected to accomodate one Y-format record of the maximum specified size and cannot subsequently be changed. Variable length records up to the maximum specified size may be stored and retrieved. Records may be stored and retrieved in any sequence. Attempts to directly retrieve (by ordinal or file address) records which have been deleted or have never been written are detected and an error code is returned. Similarly, attempts to store new records into entries which already contain records are detected and an error code is returned. Sequential retrieval causes empty entries to be bypassed; records are returned in the order they exist in the file. Sequential storage of new records implicitly generates ordinals that increase by one on each request.

NOTE: Although the rules for storage and retrieval of records conform to COBOL rules for Relative files, arbitrary use of these rules may result in unusual performance characteristics. For example, initially writing only record ordinals 1 and 50000 is permitted. Similarly, opening the file and performing two sequential retrieval requests will return only the two existing records. However, the elapsed time required to sequentially store and retrieve the record at ordinal 50000 is likely to be extremely large. In general, sparsely populated files will not show rapid performance with sequential operations.

Indexed file organization is supported only on mass storage devices and allows both sequential and random access to records. Records are uniquely identified by the content of a key field located at a fixed position in each record. Records may be randomly processed by specifying either the value of the key or the file address assigned to a record at the time it was stored. Records may also be sequentially processed in ascending order of key values. Support of alternate record keys is provided through the Multiple Index Processor which is described in a separate specification.

Library organization is supported only on mass storage devices and is intended to contain only IPL Load Modules. Neither Record-Level nor Block-Level access is available. A file having Library organization is also the only type of file which can have the Execute attribute associated with it. Files having Library organization are initially written through the OBLIGE utility program and may subsequently be opened for segment-level access, usually for execution.

NOTE: The current definition of Library organization is somewhat misleading, since it does not have the usual attributes of a file organization. The specification of library organization for a file currently is used only as a logical switch to invoke special checking of the "execute" attribute and the ring assignments.

2.9 FILE CATALOGS

A file catalog exists for each mass storage volume set and is constructed at the time the volume set is initially formatted. A catalog is implemented as a single file of indexed organization. Entries in a catalog are represented as variable length records with unique symbolic keys. The external identifier of a mass storage file (the key of the record which describes the file) is composed of three parts: owner identifier, file name and generation number. Any cataloged file at an IPL site is uniquely identified by the four-part identifier: volume set identifier, owner identifier, file name, generation number.

A more general cataloging facility has been the subject of a great deal of discussion during IPL development but insufficient design effort has been invested to permit its specification at this time. The proposed generality extends the current capability in two ways. First, a capability to catalog things other than mass storage files would be provided. Primary early candidates for this include descriptions of tape files and volume sets, mass storage volume sets and terminals on switched lines. The second extension would abandon the implicit four-part logical naming hierarchy discussed in the preceding paragraph in favor of a more general explicit N-level physical hierarchy of catalogs. The two extensions are relatively independent in that neither depends on the existence of the other.

3.0 VOLUME MANAGEMENT REQUESTS

To be supplied.

4.0 FILE MANAGEMENT REQUESTS

File Management requests assist a user in preparing files for processing and disposing of them after processing is completed. All requests except those which open a file, close a file and close a tape volume are available both through Command Language and internally. These three are available only as internal requests.

Before opening a file, an open descriptor must be allocated in any memory addressable by the user. The open request places in the open descriptor an internal file identifier and a link to the appropriate access procedure. The open descriptor is a required parameter on all explicit requests made while the file is open (open, close and close-volume File Management requests, all record-level and block-level requests).

All other File Management requests have the identifier of the File Control Block (fcbid) as a required parameter. Through Command Language, this is the LNS name of the File Control Block. The form of fcbid for internal requests is currently being reassessed. Its precise form will be defined in a subsequent update to the GDS and will either be identical to the external form (the LNS name of the File Control Block) or will be a pointer to an internal descriptor of the File Control Block.

File Management requests execute serially with respect to the requestor. The request status field, which is set before returning from each request, specifies whether the request was completed successfully or was terminated because of an error condition. Error conditions are defined in Appendix E.

4.1 FM#DEFINE_FILE

This request constructs a File Control Block and places parameters into it or merges parameters into an existing File Control Block. If the File Control Block exists, the field values transmitted through this request will be placed into the File Control Block only if the corresponding File Control Block fields contain a null value. The request and its parameters are:

FM#DEFINE_FILE (fcbname, paramblock, status)

fcbname: the LNS name of the File Control Block that is to be constructed or modified (required).

paramblock: the location of the parameter block which contains the field values for the File Control Block (required).

status: the location of the status field for this request (required).

This request is rejected if the File Control Block exists and is currently locked. The fields of the parameter block that may be transmitted to the File Control Block through this request are described in Appendix A. The SWL representation of the parameter block is described in Appendix G.

4.2 FM#CREATE_FILE

This request creates a temporary mass storage file. Null fields in the specified File Control Block are supplied default values at this time and the physical mass storage space for the file is allocated . The request and its parameters are:

FM#CREATE_FILE (fcbid, blocks, vord, rau, status)

fcbid: the identifier of the File Control Block for this file (required).

blocks: the number of blocks to be allocated (optional). The size of each block is specified in the File Control Block. If this parameter is null, an installation-supplied default value is chosen.

vord: the volume ordinal, within the volume set identified in the File Control Block, on which to restrict the space allocation (optional).

rau: The relative allocation unit, within the volume identified by the vord parameter, at which to begin the allocation (optional and cannot be specified unless the vord parameter is also specified).

status: the location of the status field for this request (required).

Use of the vord and rau parameters require the user to have increasingly detailed knowledge of the current space assignment on the specified volume set. The intended use of the two parameters is to allow users to control the placement of large permanent files on private volume sets. The intended use of the vord parameter alone is to allow users to force files, which are later to be processed simultaneously, onto separate volumes of private volume sets in order to minimize contention. When the rau parameter is supplied, physically contiguous allocation is implied and the request is rejected if sufficient contiguous space at the specified location is not available. If the rau parameter is not supplied, space is allocated according to a standard system algorithm and may or may not be physically contiguous.

4.3 FM#EXPAND_FILE

This request allocates additional space for a temporary or permanent mass storage file. The request and its parameters are:

FM#EXPAND_FILE (fcbid, blocks, vord, rau, status)

fcbid: the identifier of the File Control Block for this file (required).

blocks: the number of additional blocks to be allocate (required). The size of each block is specified in the File Control Block.

vord: the volume ordinal, within the volume set identified in the File Control Block, on which to restrict the space allocation (optional).

rau: the relative allocation unit, within the volume identified by the vord parameter, at which to begin the allocation (optional and cannot be specified unless the vord parameter is also specified).

status: the location of the status field for this request (required).

The use of the vord and rau parameters is the same as described in the FM#CREATE_FILE request. This request may only be issued by the owner of the file and only if the file is exclusively attached (if permanent) or created (if temporary) and is not currently open. Note that implicit expansion of the file will be done as needed by the system if the maximum file size and expansion increment are specified appropriately in the File Control Block. Explicit expansion is intended to accomodate the user who wishes to exercise more control over the physical placement of the file.

4.4 FM#CONTRACT_FILE

This request releases mass storage space currently allocated to a temporary or permanent file. The request and its parameters are:

FM#CONTRACT_FILE (fcbid, blocks, status)

fcbid: the identifier of the File Control Block for this file (required)

blocks: the number of blocks of currently allocated space to be returned (optional).

status: the location of the status field for this request (required).

This request is valid only for mass storage files. It may be issued only by the owner of the file and only if the file is attached for exclusive use (if permanent) or created (if temporary) and is not currently open.

If the value of the "blocks" parameter is null, unused space (beyond the highest recorded block number) is released.

4.5 FM#SAVE_FILE

This request converts a temporary mass storage file to a permanent file and records its logical and physical description in the file catalog of the appropriate volume set. The ownership of the file is permanently established at this time. The request and its parameters are:

FM#SAVE_FILE (fcbid, filename, generation, status)

fcbid: the identifier of the File Control Block for this file (required).

filename: the external name under which the file is to cataloged (optional). If filename is omitted, the system will attempt to catalog the file using the LNS name of the File Control Block.

generation: the generation number under which the file is to be cataloged (optional). If this parameter is omitted, a generation number of 0001, or a generation number one greater than the highest generation number existing for the specified name, will be assigned.

status: the location of the status field for this request. (required)

This request is valid only for temporary mass storage files and may be issued only if the file has been created and is not currently open. After successful completion of this request, the file attributes are identical to those of a permanent file which is attached for exclusive use.

4.6 FM#DELETE_FILE

This request causes a temporary or permanent mass storage file to be deleted from the system and all of its allocated space to be returned for subsequent reuse. The request and its parameters are:

FM#DELETE_FILE (fcbid, status)

fcbid: the identifier of the File Control Block for this file (required).

status: the location of the status field for this request (required).

This request may be issued only by the owner of a file and only if the file is attached for exclusive use (if permanent) or created (if temporary) and is not currently open.

4.7 FM#ATTACH_FILE

This request causes a permanent mass storage file to be attached to a job for exclusive or shared use. Intermediate operations performed include; retrieval of the cataloged description of the file, checking of the Access Control list to validate that the current user has the access rights being requested, calling the File Access Procedure associated with the file (if one is specified), verification that the requested access rights do not conflict with access rights being currently exercised by other users, storing cataloged information into the File Control Block and locking the File Control Block to prevent further modification. The request and its parameters are:

FM#ATTACH_FILE (fcbid, use, status)

fcbid: the identifier of the File Control Block for this file (required).

use: the sharing and use selection for this attachment of the file (optional). This parameter may have one of six values:

E	- Exclusive use
PI	- Protected input
PU	- Protected update
PX	- Protected execution
UI	- Unprotected input
UU	- Unprotected update

Omission of this parameter causes exclusive use to be selected.

status: the location of the status field for this request (required).

Appendix B summarizes the open uses that may be selected for each attach use and also summarizes the valid concurrent attach uses.

Three categories of information may be supplied in the File Control Block to conditionally override the corresponding cataloged values.

1. Buffering selections override the cataloged selections if no conflicts with current attach selections are detected.

2. Access-level selection overrides the cataloged selection if no conflicts with concurrent attach selections are detected, except that record level access may never be selected if block-level or segment-level access is specified in the catalog. Note, however, that the cataloged ring options may still prevent access to the data, even though the selection of a different access level is permitted.
3. FAP options in the File Control Block will be used if no FAP options are specified in the cataloged file descriotion. However, cataloged FAP options are unconditionally placed into effect if specified.

4.8 FM#DETACH FILE

This request causes a permanent mass storage file to be detached from the current job. Intermediate operations performed include: decrementing internal usage counts associated with the file, releasing internal tables associated with the file if no other jobs are attached, updating usage statistics in the cataloged description of the file and unlocking the File Control Block associated with the file in this job. The request and its parameters are:

FM#DETACH_FILE (fcbid, status)

fcbid: the identifier of the File Control Block for this file (required).

status: the location of the status field for this request (required).

This request may be issued only if the file is currently attached to this job (either by explicitly attaching a permanent mass storage file or by creating a temporary mass storage file and then saving it) and is not currently open.

4.9 FM#OPEN_FILE

This request establishes a logical connection between a file and an executing program and constructs the internal tables necessary to allow access to the data contained in the file. A file must be opened before any data transfer requests are allowea. The request and its parameters are:

FM#OPEN_FILE (fcbid, opendesc, use, labelproc, segno, pva, status)

fcbid: the identifier of the File Control Block for this file (required).

opendesc: the address of an Open Descriptor which will be used to reference the file while it is open (required). Upon completion of the open request, the Open Descriptor contains two values which specify: (a) the internal identifier of the file (relevant only to the data management system), (b) a pointer to a binding section entry which identifies the procedure to call for all requests while the file is open.

use: the use option selected for this instance of open of the file (required). This parameter may have one of file values:

I - input
O - output
E - extend
U - update
X - execution

labelproc: the entry point of a user-supplied label processing routine which is expected to be entered when labels are encountered (optional). Omission of this parameter causes the system to follow standard label processing procedures.

segno: the process segment number under which the file is to be opened (optional and has meaning only for segment-level access). If specified, the segment number must have previously been reserved. If null, a segment number will be chosen and returned in the pva field.

pva: the process virtual address to be used for segment level access (returned after successful completion of open for

segment-level access).

status: the location of the status field for this request (required).

Input usage means that the file exists and will be considered read-only while open. It is positioned at its beginning and only retrieval and positioning requests are allowed. Output usage means the file is being initially written or rewritten. It is positioned at its beginning and only data storage requests are allowed. Extend usage means that data is being appended to the file. It is positioned at end-of-recorded-data and only data storage requests are allowed. Update usage means that all requests are valid, subject only to limitations imposed by hardware or file organization. Execute usage, which may be selected only for Library file organzation, means that the program contained in the file will be executed.

Appendix B summarizes the valid open usage selections for each attach usage selection. Temporary mass storage files and non mass storage files are treated as though they are attached for exclusive use. Open usage selections must be validated by Access Control List checks (on files which have Access Control Lists).

Mass storage file may be opened after they have been created (if temporary) or attached (if permanent).

Files on magnetic tape may be opened after a File Control Block has been defined and Volume Sets have been attached to the job. Default values and/or values from standard labels are placed in the File Control Block at this time. Multiple instances of open of files on magnetic tape are not allowed.

Files which map to unit record devices may be opened after the device has been reserved and acquired and a File Control Block has been defined. Unit record device scheduling is described in the Job Management chapter of the GDS.

4.10 FM#CLOSE_FILE

This request severs the logical connection between a file and an executing program that was established when the file was opened and prevents access to the data contained in the file. The request and its parameters are:

FM#CLOSE_FILE (opendesc, disposition, status)

opendesc: the address of the open descriptor returned at the time the file was opened (required).

disposition: the disposition to be made of a tape file after the close operation is completed (optional). This parameter may have one of the four values listed below:

E - leave the volume positioned at the end of the current file.
B - leave the volume positioned at the beginning of the current file.
R - rewind the volume.
U - rewind and unload the volume.

Omission of this parameter causes the volume to be rewound. The parameter is ignored for non-tape files.

status: the location of the status field for this request (required).

4.11 FM#CLOSE_VOLUME

This request causes the current volume of a tape file to be closed and a new volume opened. If the file is opened for output or extension then labels are generated, if necessary, and exits to user-supplied label processing routines are taken. If the file is opened for input, then labels are checked and exits to user-supplied label processing routines are taken. The request and its parameters are:

FM#CLOSE_VOLUME (opendesc, disposition, status)

opendesc: the address of the open descriptor returned at the time the file was opened (required).

disposition: the disposition to be made of the current volume of the file (optional). One of four values may be specified:

E - leave the volume positioned at the end of the current file or file section.
B - leave the volume positioned at the beginning of the current file or file section.
R - rewind the volume.
U - rewind and unload the volume.

Omission of this parameter causes the volume to be rewound.

status: the location of the status field for this request. (required)

4.12 FM#PERMIT_ACCESS

This request adds a new entry or modifies an existing entry in an Access Control List associated with a permanent mass storage file and describes the access rights which are to be allowed. The request and its parameters are:

FM#PERMIT_ACCESS (fcbid, userid, accountid, shareoption, useoption, status)

fcbid: the identifier of the File Control Block for this file (required).

userid: the identification of the user for which these access rights are being established (optional). Omission of this parameter results in a null userid which is interpreted by the system as "any user".

accountid: the identification of the account for which these access rights are being established (optional). Omission of this parameter results in a null accountid which is interpreted by the system as "any account".

shareoption: the sharing options which are to be enabled for this combination of user and account for this file (required). This parameter may have any combination of the following three values:

E - exclusive use is allowed
P - protected use is allowed
U - unprotected use is allowed

useoption: the usage options which are to be enabled for this combination of user and account for this file (required). This parameter may have any combination of the following six values:

I - the file may be attached and opened for input
O - the file may be attached and opened for output
E - the file may be attached and opened for extension
U - the file may be attached and opened for update
X - the file may be attached and opened for execution

status: the location of the status field for this request (required).

This request may be issued only by the owner of the file and

only if the file is attached for exclusive use and is not currently open.

If this request results in creation of a new Access Control List entry then only the specified access rights are allowed. If an Access Control List entry already exists for this combination of user and account, then the specified access rights are "added to" the access rights currently defined.

4.13 FM#PROHIBIT_ACCESS

This request adds a new entry or modifies an existing entry in an Access Control List associated with a permanent mass storage file and describes the access rights which are to be disallowed. The request and its parameters are:

FM#PROHIBIT_ACCESS (fcbid, userid, accountid, shareoption, useoption, status)

The parameter descriptions are identical to the parameter descriptions for the FM#PERMIT_ACCESS request.

This request may be issued only by the owner of the file and only if the file is attached for exclusive use and is not currently open.

If this request results in creation of a new Access Control List entry, then its effect is to deny access to the specified user and account combination. If an Access Control List entry already exists for this combination of user and account, then the specified access rights are "subtracted from" the access rights currently defined.

4.14 FM#REDEFINE_FILE

This request permits the owner of a mass storage file to redefine selected fields of the File Control Block. The request and its parameters are:

FM#REDEFINE_FILE (fcbid, paramblock, status)

fcbid: the identifier of the File Control Block for this file (required).

paramblock: the location of the parameter block which contains the new values for the fields which are to be redefined (required).

status: the location of the status field for this request (required).

This request may be issued only by the owner of the file and only if the file is created (if temporary) or attached for exclusive use (if permanent) and is not currently open.

The parameter block is constructed as described for the FM#DEFINE_FILE request. Alterable File Control Block fields are defined in Appendix A. Non-null values in the parameter block replace the value in the corresponding field of the File Control Block if the field is defined to be alterable. If the file is permanent, its cataloged description is updated when the file is detached.

5.0 RECORD MANAGEMENT REQUESTS

Record-level access to files is provided through requests which perform six basic functions: record retrieval, record storage, record replacement, record deletion, record positioning and control selection. The requests are functionally grouped as shown below.

Record Retrieval Requests (existing records)	
RM#GET	(retrieve next record)
RM#GETP	(retrieve next partial record)
RM#GETK	(retrieve record by key or ordinal)
RM#GETD	(retrieve record by file address)
Record Storage Requests (new records):	
RM#PUT	(store next record)
RM#PUTP	(store next partial record)
RM#PUTK	(store record by key or ordinal)
Record Replacement Requests (existing records):	
RM#REPLACE	(replace record previously retrieved)
RM#REPLACEK	(replace record by key or ordinal)
RM#REPLACED	(replace record by file address)
Record Deletion Requests (existing records):	
RM#DELETE	(delete record previously retrieved)
RM#DELETEK	(delete record by key or ordinal)
RM#DELETED	(delete record by file address)
Record Positioning Requests (existing records):	
RM#FINDF	(position to first record)
RM#FINDP	(position to previous record)
RM#FINDK	(position to key or ordinal)
RM#FINDD	(position to file address)
Control Selection Requests (existing records):	
RM#LOCK	(set a record lock)
RM#UNLOCK	(clear record lock(s))

5.1 REQUEST BLOCK DESCRIPTION

Parameters to record-level requests are passed through a request block which is directly accessible by the user. Most requests also return status parameters in the request block that are in addition to the standard system request status. Fields of the request block are described below.

opendesc - This field contains a pointer to the Open Descriptor that was initialized by the FM#OPENFILE request.

bufadr - This field contains a pointer to the first byte of a record buffer which is located in memory that is directly accessible by the user. All record retrieval storage and replacement requests transfer data between the user's record buffer and internal, protected, block buffers. Direct access to the internal buffers is not allowed.

buflength - This field contains the length, in bytes, of the record buffer. For retrieval requests, this field is typically set to the maximum record size defined for the file. For storage and replacement requests, this field specifies the amount of data to be transmitted.

keyadr - This field contains a pointer to the record key and has meaning only for Indexed and Relative file organization. For Indexed files, the key consists of a byte string which identifies the record. For Relative files, the key is the ordinal of the record.

keylength - This field contains the length, in bytes, of the record key and has meaning only for one request dealing with indexed files. The RM#FINDK request permits positioning to be based on a selectable number of high order bytes of the key. All other keyed requests operate with the full key length defined for the file and ignore this parameter.

fileadrin - This field contains the input fileaddress which is used by the four "direct" requests and the record locking requests to identify a record.

compareopt - This field contains the compare option to be used by the RM#FINDK request and may contain one of three values; =, >, ≥. Appropriate setting of this field permits a file to be positioned to the first record whose key or ordinal is equal to, greater than, or greater/equal to the input key.

lockopt - This field contains the record lock option to be selected and may have one of three values:

E - an exclusive lock is selected. Others may read the record. Only the selector of this lock may modify the record until the lock is cleared.

S - a shared lock is selected. Others may read the record. No one, including the selector of this lock, may modify the record until the lock is cleared.

Null - no lock is selected

waitopt - This field contains the wait option to be selected if the selected lock option conflicts with a lock option that is currently in effect. The field may contain one of two values:

R - reject the request if the specified lock option cannot immediately be satisfied.

W - wait until the specified lock option can be satisfied before returning from the request.

delim - This field contains a user-specified one-byte data delimiter and has meaning only for files containing Y-format records. The contents of this field are copied into the corresponding field of the record header by each record storage request. The contents of the corresponding field of the record header are copied into this field by each record retrieval request. The field is intended to accomodate applications that require imbedded control information in a file and cannot use data records to describe the control information.

binflag - This field contains a one-bit binary flag and has meaning only for files containing Y-format records. The content of this field is copied into the corresponding field of the record header by each record storage request. The content of the corresponding field of the record header is copied into this field by each record retrieval request. The field is intended to accomodate applications (including Fortran formatted/unformatted I/O routines) for which the concept of binary and coded records has meaning. By convention, the value "zero" implies a coded record. The value "one" implies a binary record.

NOTE: Neither the delim nor binflag field are interpreted by system record management

procedures. The fields are merely copied between the request block and the header of Y-format records. The contents of the fields may be interpreted or ignored by the user program.

fileadrout - This field contains the file address of the record referenced by the most recent record-level access request and is set upon return from each request.

count - This field contains the transfer count associated with the preceding record storage, retrieval, or replacement request. It is an output parameter and defines the number of bytes transmitted to the user's buffer (by retrieval requests) or to the file (by storage and replacement requests).

recordflag - This field specifies whether the preceding retrieval request transmitted a complete or partial record. It is an output parameter and is set to zero to indicate that a complete record was transferred. The value "one" means that the preceding retrieval request transferred a partial record.

Appendix C summarizes the record-level access requests, the record-level request parameters, the parameters which have meaning for each request, and the requests which have meaning for each file organization and hardware type.

The request descriptions in the following sections identify the parameters that are applicable to each request but do not repeat the parameter descriptions appearing in this section. Two parameters which have not previously been described also appear in each request:

rba: This parameter is a pointer to the request block and conforms to standard IPLOS conventions for Task Services requests.

status: This parameter is a pointer to the status area for the request and also conforms to standard IPLOS conventions for Task Services requests.

All other applicable parameters are contained in the request block identified by "rba".

Record management requests execute serially with respect to the requestor. The request status field, which is set before returning from each request, specifies whether the request was completed successfully or was terminated because of an error condition. Error conditions are defined in Appendix E.

5.2 PROCESSING SHARED FILES

When a permanent mass storage file is simultaneously open more than once, and any form of data modification is allowed, facilities are provided to assist in coordination of processing through the multiple instances of open.

Proper selection of the lockopt and waitopt parameters on record storage and retrieval requests and/or proper use of the RM#LOCK and RM#UNLOCK requests permit updates to a record or set of records to be controlled and properly sequenced.

The type of lock to be placed on a record is specified by the lockopt parameter. A shared lock allows the record to be retrieved through any instance of open but prevents the record from being modified. An exclusive lock allows the record to be retrieved through any instance of open and also allows it to be modified through the current instance of open.

The waitopt parameter specifies the type of action to be taken if a specified lock option conflicts with a lock option currently in effect through another instance of open (a conflict is an attempt to set an exclusive lock if any lock is currently in effect or an attempt to set a shared lock if an exclusive lock is in effect). If the wait option is R, the request is rejected if the specified lock option cannot be selected. If the wait option is W, the request is queued and the requestor will wait until the specified lock option can be selected unless queueing the request would generate a deadlock condition.

A deadlock condition exists if the program which has the specified record currently locked it also queued waiting (directly or indirectly) for another record that is currently locked through this instance of open. The action taken when a deadlock is detected is to reject the request. A user should, therefore, lock all records of a set which require "simultaneous" update before performing any of the updates. A user should also be prepared to unlock all records in the set and reinitiate the entire locking sequence in case a deadlock is detected by the system.

The valid combinations of current lock selection and existing lock selections are shown below:

		Current Selection		
		None	Shared	Exclusive
Existing Selection	None	OK	OK	OK
	Shared	OK	OK	-
	Exclusive	OK	-	-

5.3 RM#GET

This request transfers the "next" existing record or partial record from a file to a buffer. The request and its input parameters are:

```
    RM#GET  (rba,  status,  opendesc, bufadr, buflength, lockopt,
       waitopt)
```

Output parameters are:

```
    (delim, binflag, fileadrout, count, recordflag)
```

The request has meaning for all file organizations and all hardware types for which data retrieval is defined. Successive RM#GET requests retrieve records in an order determined by the organization of the file. Records in sequentially organized files are retrieved in order of their physical placement. Records in files having Indexed organization are retrieved in order of ascending value of the record key. Records in files having Relative organization are retrieved in order of ascending value of record ordinal (empty entries are skipped).

All transfers begin at a record boundary and are terminated by the smaller of record length or buffer length.

5.4 RM#GETP

The request transfers the remainder or partial remainder of a record from a file to a buffer. The request and its input parameters are:

RM#GETP (rba, status, opendesc, bufadr, buflength)

Output parameters are:

(count, record flag)

The request is valid only for sequentially organized files and is intended to continue the retrieval of a record which was partially retrieved by a previous request. Transfers begin at the location, within the record, following the end of the preceding partial transfer and are terminated by the smaller of remaining record length or buffer length. The request is not valid unless the preceding request caused the partial record flag to be set.

NOTE: RM#GETP cannot begin a transfer at a record boundary and cannot cross a record boundary. It can only be used to complete the transfer of a record that was started with one of the other retrieval requests.

5.5 RM#GETK

This request transfers a record, identified by its key value or ordinal value, from a file to a buffer. The request and its input parameters are:

RM#GETK (rba, status, opendesc, bufadr, buflength, keyadr, lockopt, waitopt)

Output parameters are:

(delim, binflag, fileadrout, count, recordflag)

The request is valid only for files having Indexed or Relative organization. For Indexed organization, the keyadr parameter points to the first byte of the symbolic key of the record to be retrieved. For Relative organization, the keyadr parameter points to the ordinal of the record to be retrieved. All transfers begin at a record boundary and are terminated by the smaller of buffer length or record length.

The request is rejected if a record with the specified key does not exist in the file.

5.6 RM#GETD

This request transfers a record, identified by its file address, from a file to a buffer. The request and its input parameters are:

```
    RM#GETD (rba, status, opendesc, bufadr, buflength, fileadrin,
      lockopt, waitopt)
```

Output parameters are:

```
    (delim, binflag, fileadrout, count, recordflag)
```

The request is valid only for mass storage files. All transfers begin at a record boundary and are terminated by the smaller of buffer length or record length. The input file address will typically have been saved from the time the record was stored in the file.

The request is rejected if the specified record does not exist in the file.

5.7 RM#PUT

This request transfers a record, or completes the transfer of a record, from a buffer to the "next" location in a file. The request and its input parameters are:

```
    RM#PUT (rba, status, opendesc, bufadr, buflength, lockopt,
      delim, binflag)
```

Output parameters are

```
    (fileadrout, count)
```

If the file contains fixed length records, the record is blank filled to the correct length if necessary and the count parameter shows the actual length transferred to the file. If the file has Indexed organization, the key of the record must be greater than the key of the preceding record. If the file has Relative organization, an ordinal one greater than the ordinal of the preceding record is assigned.

This request unconditionally marks the end of recorded data on the file. Records may not be retrieved beyond this point.

5.8 RM#PUTP

This request transfers a partial record from a buffer to a file. The request and its input parameters are:

RM#PUTP (rba, status, opendesc, bufadr, buflength, delim, binflag)

Output parameters are:

(fileadrout, count)

The request is valid only for sequentially organized files and either begins or continues the storage of a new record. If a new record is being started (the preceding request was not RM#PUTP) the delim and binflag parameters have meaning. If a new record is being continued (the preceding request was RM#PUTP) the delim and binflag parameters are ignored.

The partial record requests are intended to accomodate applications which are unable to transfer complete records, usually because the records are extremely long. These records can be written by use of a sequence of RM#PUTP requests followed by RM#PUT to terminate the record. The records can subsequently be read by RM#GET or RM#GETD followed by a sequence of RM#GETP requests (if necessary) to complete the retrieval.

5.9 RM#PUTK

This request transfers a record, identified by its key value or ordinal value, from a buffer to a file. The request and its input parameters are:

RM#PUTK (rba, status, opendesc, bufadr, buflength, keyadr, lockopt, delim, binflag)

Output parameters are:

(filedrout, count)

The request is valid only for files having Indexed or Relative organization. For indexed organization, the key value is obtained directly from the record buffer and the keyadr parameter is ignored. For relative organization, the keyadr parameter points to the ordinal of the record to be stored.

The request is rejected if a record with an identical key already exists in the file.

5.10 RM#REPLACE

This request replaces the record obtained through an immediately preceding retrieval request. The request and its input parameters are:

RM#REPLACE (rba, status, opendesc, bufadr, buflength)

Output parameters are:

(fileadrout, count)

The request is valid only for mass storage files. For Sequential organization, the length of the record cannot be changed and the file cannot contain spanned records (of either S or Y format). For Indexed organization, the value of the key may not be changed. The request is rejected if it is preceded by any request other than RM#GET, RM#GETK or RM#GETD.

5.11 RM#REPLACEK

This request replaces a record, identified by its key value or ordinal value, in a file. The request and its input parameters are:

RMREPKEY (rba, status, opendesc, bufadr, buflength, keyadr)

Output parameters are

(fileadrout, count)

The request is valid only for files with Indexed or Relative organization. For Indexed organization, the key value is obtained directly from the record buffer and the keyadr parameter is ignored. For Relative organization, the keyadr parameter points to the ordinal of the record to be replaced.

The request is rejected if a record with the specified key does not exist in the file.

5.12 RM#REPLACED

This request replaces a record, identified by its file address, in a file. The request and its input parameters are:

RM#REPLACED (rba, status, opendesc, bufadr, buflength, fileadrin)

Output parameters are:

(fileadrout, count)

The request is valid only for mass storage files. For Sequential file organization, the length of the record cannot be changed and the file cannot contain spanned records (either S or Y format). For Indexed organization, the value of the key may not be changed.

The request is rejected if the specified record does not exist in the file.

5.13 RM#DELETE

This request deletes the record obtained through an immediately preceding retrieval request. The request and its input parameters are:

RM#DELETE (rba, status, opendesc)

The output parameter is:

(fileadrout)

The request is valid only for mass storage files and is not valid for sequentially organized files unless they contain Y-format records. The request is rejected if it is preceded by any request other than RM#GET, RM#GETK or RM#GETD.

5.14 RM#DELETEK

This request deletes a record, identified by its key value or ordinal value, from a file. The request and its input parameters are:

```
    RM#DELETEK (rba, status, opendesc, keyadr)
```

The output parameter is:

```
    (fileadrout)
```

The request is valid only for files with Indexed or Relative organization and is rejected if the specified record does not exist in the file.

5.15 RM#DELETED

This request deletes a record, identified by its file address, from a file. The request and its input parameters are:

```
    RM#DELETED (rba, status, opendesc, fileadrin)
```

The output parameter is:

```
    (fileadrout)
```

The request is valid only for mass storage files and is not valid for sequentially organized files unless they contain Y-format records. The request is rejected if the specified record does not exist in the file.

5.16 RM#FINDF

This request positions a file to the first data record. The request and its input parameters are:

RM#FINDF (rba, status, opendesc)

The output parameter is:

(fileadrout)

The request is valid for all file organizations and all hardware types for which positioning is defined.

5.17 RM#FINDP

The request positions a file to the "preceding" record. The request and its input parameters are:

RM#FINDP (rba, status, opendesc)

The output parameter is:

(fileadrout)

The request is valid only for sequentially organized files and hardware types for which positioning is defined. The request exists solely to satisfy a Fortran requirement and reasonable performance should not be expected unless the file contains format F or U records.

5.18 RM#FINDK

This request positions a file to a record whose key value or ordinal value satisfies a specified search criterion. The request and its input parameters are:

RM#FINDK (rba, status, opendesc, keyadr, keylength, compareopt)

The output parameter is:

(fileadrout)

The request is valid only for files with Indexed or Relative organization. For Indexed organization, the keyadr parameter points to the first byte of the symbolic key of the record. The keylength parameter specifies the number of high-order bytes of the key on which the comparison is to be made. For Relative organization, the keyadr parameter points to the ordinal of the specified record and the keylength parameter is ignored. For either organization, the compareopt parameter specifies the search criterion. Three values are defined:

- = - position to the record whose key value equals the input key value.
- > - position to the first record whose key value is greater than the input key value.
- ≥ - position to the first record whose key value is greater than or equal to the input key value.

The request is rejected if no record satisfying the specified search criterion can be found.

5.19 RM#FINDD

This request positions a file to a record identified by its file address. The request and its input parameters are:

RM#FINDD (rba, status, fileadrin)

The output parameter is:

(fileadrout)

The request is valid only for mass storage files and is rejected if the specified record does not exist in the file.

5.20 RM#LOCK

This request sets a shared or exclusive lock on a specified record. The request and its input parameters are:

RM#LOCK (rba, status, opendesc, fileadrin, lockopt, waitopt)

No output parameters are defined.

The request is valid onlv for mass storage files and is intended to permit concurrent users of the file to coordinate their accesses to data. The effect of this request is to place a lock on a record (identified by the fileadrin parameter) through an instance of open (identified by the opendesc parameter).

5.21 RM#UNLOCK

This request clears a lock on a record that was previously set through this instance of open. The request and its parameters are:

RM#UNLOCK (rba, status, opendesc, fileadrin)

No output parameters are defined.

The request is a companion to the RM#LOCK request. If the fileadrin parameter is not null, the lock is cleared on the specified record. If the fileadrin parameter is null, the locks are cleared on all records currently locked through this instance of open.

6.0 BLOCK MANAGEMENT REQUESTS

Block-level access to files is provided through requests which perform four basic functions: block retrieval, block storage, block positioning and control selection. The requests are functionally grouped as shown below:

Block Retrieval Requests	
BM#READ	(retrieve next block)
BM#READD	(retrieve block, direct)
Block Storage Requests	
BM#WRITE	(store next block)
BM#WRITED	(store block, direct)
Block Positioning Requests	
BM#POINTF	(position to first block)
BM#POINTL	(position to last block)
BM#POINTP	(position to preceding block)
Control Selection Requests	
BM#READSTATUS	(retrieve extended status)
BM#SELECT	(transmit device-dependent function)
BM#CANCEL	(cancel outstanding request)
BM#LOCK	(set a block lock)
BM#UNLOCK	(clear block lock(s))

6.1 REQUEST BLOCK DESCRIPTION

Parameters to block-level requests are passed through a request block which is directly accessible by the user. Most requests also return status parameters in the request block that are in addition to the standard system request status. Fields of the request block are described below:

opendesc - This field contains a pointer to the Open Descriptor that was intialized by the FM#OPEN_FILE request.

ecbptr - This field contains a pointer to the Event Control Block associated with this request. Use of the Event Control Block for monitoring the progress of asynchronous requests is discussed in section 6.2.

bufadr - This field contains a pointer to the first byte of a block buffer which is located in memory that is directly accessible by the requestor.

buflength - This field contains the length, in bytes, of the block buffer. For retrieval requests, this field is typically set to the maximum block size defined for the file. For storage requests, this field specifies the amount of data to be transmitted.

blocknum - This field contains the block number at which a direct transfer is to begin. It has meaning only for mass storage files and is interpreted as a relative block number within the file. The first block of a file is numbered one.

function - This field contains a device-dependent function and has meaning only for the BM#SELECT request.

lockopt - This field contains the block lock option to be selected and has meaning only for the BM#LOCK request. One of two values may be specified:

- E - an exclusive lock is selected. Others may read the block. Only the selector of this lock may write the block until the lock is cleared.
- S - a shared lock is selected. Others may read the block. No one, including the selector of this lock, may write the block until the lock is cleared.

waitopt - This field contains the wait option to be selected if the specified lock option conflicts with a lock option that is currently in effect. One of two values may be specified:

- R - reject the request if the specified lock option cannot immediately be satisfied.
- W - wait until the specified lock option can be satisfied before returning from the request.

count - This field contains the transfer count associated with the request. It is an output parameter and defines the number of bytes transmitted to the buffer or to the file. The field is cleared when the request is issued and is set when the request is completed.

response - This field contains a summary of responses received since the request associated with this request block was issued. The field is cleared when the request is issued and is updated as responses are received.

Appendix D summarizes the block-level requests, the block-level request parameters, the parameters which have meaning for each request, and the requests which have meaning for each open usage and hardware type.

The request descriptions in the following sections identify the parameters that are applicable to each request but do not repeat the parameter descriptions appearing in this section. Two parameters which have not been previously described also appear in each request.

rba - This parameter is a pointer to the request block and conforms to standard IPLOS conventions for Task Services requests.

status - Ths parameter is a pointer to the status area for the request and also conforms to standard IPLOS conventions for Task Services requests.

All other applicable parameters are contained in the request block identified by "rba".

6.2 REQUEST MONITORING

Block Management requests may be executed synchronously with the requestor (the request is completed before control is returned) or asynchronously with the requestor (control is returned before the request is completed).

The status of a synchronous request is available in the status field identified for the request. The status field also specifies whether the request has either been completed normally or has been rejected because of an abnormal condition. In the latter case, the reason for rejecting the request is also specified.

Asynchronous requests are executed in parallel with the requestor. At the time control is returned, the request status field specifies whether the request was accepted and initiated or was rejected. The status of the operation described by the request is automatically returned by the system in the "response" field of the original request block after the asynchronous operation is complete.

A requestor may use the Program Management event monitoring requests listed below to keep informed of the progress of an asynchronous request:

```
PM#ATTACH_PROCEDURE
PM#DETACH_PROCEDURE
PM#ENABLE_EVENT
PM#DISABLE_EVENT
PM#STATUS_EVENT
PM#WAIT_EVENT
PM#WAIT_CLEAR_EVENT
PM#CLEAR_EVENT
```

By defining an Event Control Block and placing its address in the Block Management request block, the requestor, in effect, associates an event with a response from the asynchronous request. If tha ecbptr parameter is not null, Block Management procedures "cause" the event when responses are received. If an interrupt procedure is attached to the event, it is executed at the time the response is received (unless, of course, the requestor has disabled the event or in some other way, deferred its execution).

NOTE: A Block Management request block serves both as a parameter area for requests to the system and as a status area for responses from the system. Therefore, it is "active" for the life of the request and must not be reused until a final response is received.

6.3 RESPONSE FORMAT

Altnough the exact response format cannot be specified until considerably more design work has occurred, the intent and partial content of responses are discussed in the following paragraphs:

For most asynchronous requests, a single response is returned at the time the request is successfully completed or abnormaly terminated. When a request is accepted for execution, the response and count fields of the request block are cleared. As the request is executed, intermediate responses from the peripheral system are summarized and edited. When a final response is received, the accumulated transfer count and edited response are routed back to the job from which the request was issued. The request block is updated and the appropriate event is caused, if necessary.

For some asynchronous requests, intermediate responses may be received in addition to the final response described above. One example of this occurs when a multiple block transfer request is issued to the peripheral system as more than one request, possibly because of a physical discontinuity underneath a logicaly contiguous file. Further design work on communication facilities, particularly the capabilities of front-end communication concentrators, is expected to identify other examples of multiple responses generated by a single request. From the requestor's point of view, implications of multiple responses include:

1. the count field of the request block may be incremented more than once during the life of a request:

2. the response field represents the logical sum of all intermediate responses that have occurred since the request was issued.

3. the associated event is "caused" each time a response is received at the job level (this is not necessarily every time a response is received from the peripheral system).

The exact format of the response field is not yet defined but is intended to include boolean values grouped into categories as shown below. The categories and examples under each category will be updated as more information is known.

General Status

- a final response (has) (has not) been received
- intermediate responses (have) (have not) been received
- error recovery (has) (has not) been attempted
- the request was (successful) (abnormally terminated)

Intermediate Response Type

(to be defined)

Abnormal Termination Type

- unrecoverable transfer errors
  - read error
  - write error
  - lost data
- inoperable hardware errors
  - device not ready
  - device busy
  - address error
- temporary error conditions
  - device temporarily reserved
  - a previous abnormal termination affects this request

Attention Conditions

- partial block transfer occurred
- multiple block transfe- occurred
- end of recorded data encountered
- end of physical medium encountered

Device-Dependent Status

(to be defined)

6.4 PROCESSING SHARED FILES

When a permanent mass storage file is simultaneously open more than once, and any form of data modification is allowed, facilities are provided to assist in coordination of processing through the multiple instances of open.

Proper selection of the lockopt and waitopt parameters and proper use of the BM#LOCK and BM#UNLOCK requests permit updates to a block or a set of blocks to be properly sequenced.

The type of lock to be placed on a block is specified by the lockopt parameter. A shared lock allows the block to be retrieved through any instance of open but prevents the block from being modified. An exclusive lock allows the block to be retrieved through any instance of open and also allows it to be modified through the current instance of open.

The waitopt parameter specifies the type of action to be taken if a specified lock option conflicts with a lock option currently in effect through another instance of open (lock conflicts are described in section 5.2). If the wait option is R, the BM#LOCK request is rejected if the specified lock option cannot be selected. If the wait option is W, the request is queued and the requestor will wait until the specified lock option can be satisfied unless queueing the request would generate a deadlock condition.

A deadlock condition exists if the program which has the specified block currently locked is also queued waiting (directly or indirectly) for another block that is currently locked through this instance of open. The action taken when a deadlock is detected is to reject the request. The requestor should, therefore, lock all blocks of a set which require "simultaneous" update before retrieving the blocks and performing the updates. The requestor should also be prepared to unlock all blocks in the set and reinitiate the entire locking sequence in case a deadlock is detected by the system.

6.5 ALTERNATE USE OF FILES BY SYSTEM PROGRAMS

A file opened for block-level access is defined internally by a number of tables which describe its physical location, a few of its logical properties (such as block size and allocated/recorded boundaries) and which permit data transfer requests to be made to it. Although the traditional meaning of a file implies that it is a repository for data, block-level file descriptions are used another way by some operating system components. This alternate use of files is outlined in the following paragraphs since it is potentially applicable to subsequent development of on-line diagnostic/maintenance software, message control systems and other system software yet to be defined.

An important property of a block-level file description is that it provides the capabilities of a more general message router within an IPL configuration. Requests to transmit and receive data may be issued by IPL programs, either by Read/Write requests or by Status/Select requests. The requests are routed to the appropriate location and responses are routed back to the requestor by standard software conventions. Since some operating system components, File Management procedures in particular, need to communicate directly with components of the peripheral system before user-level file descriptions exist, an alternate use of block-level file descriptions is defined.

When the system is deadstarted, block-level file descriptions are generated for all first-level components of the peripheral system which have the capability to transmit or receive data or control information. These components include all local devices (disk drives, tape drives, unit record devices and probably ports on communication controllers), all programmable controllers and, possibly, all hard-wired controllers. These block-level "files" are, essentially, opened for update and Open Descriptors are tabled in protected system memory. An operation equivalent to setting the recordring, readring and writering parameters of the File Control Block to the value of one is performed and the blockring parameter is logically set to the execution ring of Task Services. This permits Task Services procedures to issue block-level requests to the "files" through protected Open Descriptors but prohibits any programs in higher rings from accessing the "files". When a File Management procedure, for example, needs to perform operations such as reading/writing labels on storage volumes, retrieving catalog descriptions when attaching volume sets, or loading a forms control matrix into a programmable printer controller, the requests to perform these operations are made through the standard block management interface to the appropriate "file".

A number of questions have been raised concerning the ability of the current design to support a front-end communication concentrator that services large numbers of terminals. The questions usually include the observations that these applications typically require some sort of shortcut through the layers of system I/O software, some way to transmit large blocks of messages between central memory and the communication concentrator, some way to manage queues of messages in central memory and some way to imbed routing information within messages. Potential guidelines for implementing such a facility within the current operating system structure are discussed below.

First, the central software could be appropriately implemented as a protected subsystem which communicates directly with the software in the front-end concentrator by standard block management requests to a "file" of the type described in the preceding paragraphs. Arbitrarily large blocks of messages of arbitrary format could be exchanged according to mutually agreeable conventions. Second, the user interface to the protected subsystem could be either a file-type interface, where the subsystem is treated as a FAP, or a message-type interface where the user is consciously aware that he is dealing with messages and a message subsystem rather than with records and files. The current structure allows either approach to be taken. Third, the queue maintenance could be developed as special subsystem logic with queues maintained in global, protected memory segments which are accessible only at the subsystem level and below.

Similarly, diagnostic and maintenance subsystems could be implemented as protected subsystems which communicate directly with components of the peripheral systems through standard block management requests to special "files". Obviously, the "files" should either be partitioned from those in use for normal operation or great care should be taken to avoid disruption to concurrent users. The mechanisms for removing and returning these "files" from/to normal use are expected to be defined as the designs of both the operating system and the maintenance subsystem continue to be developed.

6.6 BM#READ

This request transfers data from the next block of a file to a buffer. The request and its parameters are:

BM#READ (rba, status, opendesc, ecbptr, bufadr, buflength, count, response)

If the buffer length is less than the length of the block, excess data in the block is not transferred. The response field is set to indicate that a partial block transfer ocurred and the count field is set to the actual transfer size (the buffer length if the request completed normally).

If the buffer length is greater than the length of the block, the action taken is hardware-dependent.

1) For mass storage files, data is transferred from consecutive blocks until the buffe~ is filled to capacity. The response field is set to indicate that multiple blocks were transferred and the count field is set to the actual transfer size.

2) For non-mass storage files, only a single block is transferred. The response field is set to indicate this fact and the count field is set to the actual transfer size.

6.7 BM#READD

This request transfers data from a specific block of a mass storage file to a buffer. The request and its parameters are:

```
BM#READD (rba, status, opendesc, ecbptr,  bufadr,  buflength,
   blocknum, count, response)
```

The request is valid only for mass storage files. The rules for partial and multiple block transfers are identical to those described for the BM#READ request.

6.8 BM#WRITE

This request transfers data from a buffer to the next block of a file. The request and its parameters are:

```
BM#WRITE (rba, status, opendesc, ecbptr,  bufadr,  buflength,
   count, response)
```

The buffer length specifies the amount of data to be transferred and has hardware-dependent implications if it exceeds the maximum block length specified for the file.

1) For mass storage files, data is transferred from the buffer to consecutive blocks in the file until the buffer is emptied. The response field is set to show that multiple blocks were transferred and the count field is set to the actual transfer size.

2) For non-mass storage files, the request is rejected and the request status field is set to indicate the reason.

If the buffer length is less than or equal to the maximum block length specified for the file, then only a single block is transferred for files on all hardware types.

The BM#WRITE request unconditionally marks the end of recorded data on the file. Data may not be retrieved beyond this point.

6.9 BM#WRITED

This request transfers data from a buffer to a specific block of a mass storage file. The request and its parameters are:

```
    BM#WRITED (rba, status, opendesc, ecbptr, bufadr, buflength,
      blocknum, count, response)
```

The request is valid only for mass storage files, The rules for single and multiple block transfers are identical to those described for the BM#WRITE request.

The BM#WRITED request marks the end of recorded data on the file only if the transfer terminates beyond the existing end of recorded data.

6.10 BM#POINTF

This request positions a file to the first data block. The request and its parameters are:

```
    BM#POINTF (rba, status, opendesc, ecbptr, response)
```

The request is valid for all hardware types for which positioning is defined. After successful completion of this request, the file is positioned so that a BM#READ request will retrieve the first data block. For multivolume tape files, the file is positioned to read the first data block of the current file section.

If the file is on mass storage, this request results only in the updating of an internal table. If the file is on magnetic tape, this request results in positioning of the physical medium.

6.11 BM#POINTL

This request positions a file to the last data block. The request and its parameters are:

BM#POINTL (rba, status, opendesc, ecbptr, response)

This request is valid for all hardware types for which positioning is defined. After successful completion of the request, the file is positoned so that a BM#READ request will return "end of recorded data" status. A BM#WRITE request will extend the file. For multivolume tape files, the file is positioned to the end of the current file section.

If the file is on mass storage, this request results only in the updating of an internal table. If the file is on magnetic tape, this request results in positionig of the physical medium.

6.12 BM#POINTP

This request positions a file to the "preceding" data block. The request and its parameters are:

BM#POINTP (rba, status, opendesc, ecbptr, response)

This request is valid for all hardware types for which positioning is defined. After successful completion of this request, the file is positioned in front of the last data block accessed by the last preceding storage, retrieval or positioning request.

If the file is on mass storage, this request results only in the updating of an internal table. If the file is on magnetic tape, this request results in the tape being backspaced one physical record. Multivolume tape files cannot be backspaced out of the current file section.

6.13 BM#READSTATUS

This request retrieves a variable amount of device-dependent status from the controller associated with the device on which the specified file resides. The request and its parameters are:

BM#READSTATUS (rba, status, opendesc, ecbptr, bufadr, buflength, count, response)

This request is intended for use by diagnostic procedures, error recovery procedures, logging procedures or others which required device-dependent interaction with the peripheral system.

The size limitations and data format of the buffer are device-dependent and will be defined as the appropriate controller specifications are developed.

6.14 BM#SELECT

This request transmits device-dependent functional data to the controller associated with the device on which the specified file resides. The request and its parameters are:

BM#SELECT (rba, status, opendesc, ecbptr, bufadr, buflength, blocknum, function, count, response)

This request is intended for use by diagnostic procedures, error recovery procedures, logging procedures or others which require device-dependent interaction with the peripheral system.

The size limitations and data format of the buffer, the use of the blocknum parameter and the meanings of the function parameter are device-dependent and will be defined as IPL peripheral specifications become available.

NOTE: A cursory examination of a few existing controller specifications produced a list of over fifty functions that various controllers can recognize. Some functions can legitimately be made available to user programs that have non-shareable files (such as tapes) open for block-level access. Other functions can be made available only to authorized privileged system programs since their misuse could lay waste to the system. As IPL peripheral specifications become avilable and their capabilities become known, the availability of device-dependent functions to various types of procedures will be clarified.

6.15 BM#CANCEL

This request cancels a previously issued, asynchronous, block-level request. The request and its parameters are:

BM#CANCEL (rba, status)

The request block address identifies the previously issued request that is to be cancelled. The intent of this request is to allow programs to terminate requests that appear to have gone astray, possibly because a reponse has not been received for an excessively long time or because an abnormal response from another request indicates that something in the peripheral system is misbehaving. The limitations of this request and clarification of other uses will be defined as the design of all related system components becomes better known.

6.16 BM#LOCK

This request sets a shared or exclusive lock on a specified block of a file. The request and its parameters are:

BM#LOCK (rba, status, opendesc, blocknum, lockopt, waitopt)

The request is valid only for mass storage files and is intended to permit concurrent users of the file to coordinate their accesses to data. The effect of this request is to place a lock on a specific block (identified by the blocknum parameter) through an instance of open (identified by the opendesc parameter).

6.17 BM#UNLOCK

This request clears a lock on a block that was previously set through this instance of open. The request and its parameters are:

```
BM#UNLOCK (rba, status, opendesc, blocknum)
```

The request is a companion to the BM#LOCK request. If the blocknum parameter is not null, the lock is cleared on a specific block. If the blocknum parameter is null, the locks are cleared on all blocks locked through this instance of open.

7.0 APPENDIX A - FIELDS OF A FILE CONTROL BLOCK

Summary of User-Accessible Fields							Relevance by Hardware Type						
							Mass Storage File Org				Mag. Tape		Unit Rec & Term
Category	Field	Type	Size	Notes	Defaults	RDF.	IND	REL	SEQ	LIB	STD. LABELS	NS/NO LABELS	
General File Definition Fields	location	char	31	VSCB/DCB Name	System Volume Set	—	*	*	*	*	*	*	*
	owner	char	31	for mass storage	User ID from JCB	—	*	*	*	*	—	—	—
	filename	char	31	external name	FCB Name	—	*	*	*	*	*	—	—
	generation	char	4	0001-9999	variable	—	*	*	*	*	*	—	—
	expiration	char	5	YYDDD	Current Date	*	*	*	*	*	*	—	—
	fapname	char	31	Subsystem Name	none	*	*	*	*	*	*	*	*
	fapentry	char	31	Entry Point Name	none	*	*	*	*	*	*	*	*
	fapring	integer	1	Execution Ring 1-15	variable	*	*	*	*	*	*	*	*
	fapparam	pointer	6	FAP parameters	none	—	*	*	*	*	*	*	*
	acclevel	char	1	R/B/S (Rec/Blk/Seg)	R (record-level)	—	*	*	*	S	R,B	R,B	R,B
	fileorg	char	1	I/R/S/L	S (sequential)	—	I	R	S	L	S	S	S
	blocksize	integer	2	max. number bytes	System Page Size	—	*	*	*	*	*	*	*
	buffermode	char	1	E/I (Explicit/Impl.)	E (explicit)	*	*	*	*	I	E	E	E
	buffercount	integer	1	only if Explicit	variable	*	*	*	*	—	*	*	*
	recordring	integer	1	1-15	variable	*	*	*	*	—	*	*	*
	blockring	integer	1	1-15	variable	*	*	*	*	—	*	*	*
	writering	integer	1	1-15	variable	*	*	*	*	*	—	—	—
	readring	integer	1	1-15	variable	*	*	*	*	*	—	—	—
Record-Level Access Fields	recformat	char	1	F/D/S/U/Y	Y (system variable)	—	Y	Y	*	—	*	*	Y,U
	recsize	integer	2	max. number bytes	variable	—	*	*	*	—	*	*	*
	keyloc	integer	2	relative byte loc.	none	—	*	—	—	—	—	—	—
	keysize	integer	1	number of bytes	none	—	*	—	—	—	—	—	—
	loading	integer	1	Percent capacity	100 percent	—	*	—	—	—	—	—	—
	compression	char	1	Y/N (Yes/No)	N (no compression)	—	—	—	*	—	*	*	*
	spanopt	char	1	Y/N (Yes/No)	N (no spanning)	—	—	—	*	—	*	*	*
Mass Storage Fields	filesize	integer	4	max. number blocks	variable	*	*	*	*	*	—	—	—
	expansion	integer	4	number of blocks	variable	*	*	*	*	*	—	—	—
	vord	integer	1	ordinal in VSCB	FF16 (any volume)	*	*	*	*	*	—	—	—
	verify	char	1	Y/N (Yes/No)	N (no verification)	*	*	*	*	*	—	—	—
	userdata	pointer	6	optional	none	—	*	*	*	*	—	—	—
	fapdata	pointer	6	only if FAP exists	none	—	*	*	*	*	—	—	—
Mag. Tape Fields	sequence	char	4	0001-9999	none	—	—	—	—	—	*	—	—
	section	char	4	0001-9999	none	—	—	—	—	—	*	—	—
	hdropt	char	1	Y/N (Yes/No)	Y (use blockhdr)	—	—	—	—	—	*	*	—
	bsnopt	char	1	Y/N (Yes/No)	Y (use seq. number)	—	—	—	—	—	*	*	—
Unit Record & Terminal Fields	linesize	integer	2	max. number bytes	variable	—	—	—	—	—	—	—	*
	pagesize	integer	2	max. number lines	variable	—	—	—	—	—	—	—	*
	formcntl	integer	1	matrix number	FF16 (use standard)	—	—	—	—	—	—	—	*
	papercntl	integer	1	matrix number	FF16 (use standard)	—	—	—	—	—	—	—	*
	linedelim	char	1	for keyboard devices	variable	—	—	—	—	—	—	—	*
	backspace	char	1	for keyboard devices	variable	—	—	—	—	—	—	—	*
	msgdelim	char	1	for keyboard devices	variable	—	—	—	—	—	—	—	*

↑ Number of Bytes

↑ The field can be redefined

LEGEND: Null values for all fields are binary zero

* - The field has meaning

— - The field does not have meaning

other - Only the specified values are allowed

location - For tape and mass storage files, this field contains the LNS name of the Volume Set Control Block which describes the volume set on which this file resides. For unit record files, this field contains the LNS name of the Device Control Block which describes the device on which this file resides. The default value is the name of the System Volume Set.

owner - This field has meaning only for mass storage files and contains the user identifier under which the file is cataloged. The default value is the user identifier contained in the Job Control Block of the current job.

filename - This field contains the external name under which this file is cataloged or labeled. For mass storage files, the field may contain any character string up to 31 characters in length. For standard labeled tape files, the name may be up to 17 characters in length. The field has no meaning for tapes with nonstandard or no labels or for terminals and unit record devices. If the field is null and a name is required, the LNS name of the File Control Block is supplied as default.

generation - This field contains the four-byte generation number under which this file is cataloged or labeled and has meaning only for mass storage files and standard labeled tape files.

expiration - This field contains the five byte expiration date for this file (in the form YYDDD) and has meaning only for mass storage files and standard labeled tape files. If the field is null and a date is required, the current date is supplied as default.

fapname - This field contains the name of a File Access Procedure which is expected to run as a protected subsystem (the Data Base Control System, for example). If this field contains a name, linkage to the FAP will be made only through the Job Gate Table. If this field is null, FAP linkage will be attempted using the standard Loader search rules. There is no default value for this field.

fapentry - This field contains the name of the entry point to a File Access Procedure. If this field is null, no FAP linkage will be attempted. There is no default value for this field.

fapring - This field specifies the ring number at which the File

Access Procedure is expected to execute. If FAP linkage is attempted and the actual ring of execution is determined to be higher than this, the request is terminated abnormally. FAP linkage will rot occur and the file cannot be accessed. If the field is null, the actual ring obtained from the FAP linkage procedure is used.

fapparam - This field points to a parameter block through which the File Access Procedure expects to receive parameters from the user. The user may construct the parameter block in any accessible memory and is responsible for setting this field appropriately before issuing file management requests (except to define the file). There is no default value for this field.

acclevel - This field defines the access level to be selected when the file is opened for processing. Three values are defined:

R - Record-level access
B - Block-level access
S - Segment-level access (mass storage files only)

If the field is null, record-level access is selected as default.

fileorg - This field defines the organization of the file and may have one of four values:

I - Indexed organization
R - Relative organization
S - Sequential orgnization
L - Library organization (for load modules only)

If the field is null, Sequential organization is selected as default.

blocksize - This field defines the maximum block size (in bytes) for this file. If the field is null, the system page size is supplied as default.

buffermode - This field defines the buffering mode for the file and has meaning only if record-level access is also selected. The allowable values are:

E - Explicit buffering is to be used
I - Implicit buffering is to be used

Explicit buffering means that the record-level access procedures will use block-level access to the data. Implicit buffering means that the record-level access procedures will use Segment-Level access to the data. If the field is null, explicit buffering is selected as default.

buffercount - This field defines the number of buffers to be used when the file is processed and also establishes the maximum number of simultaneously active block-level requests for any instance of open. The field does not have meaning for files which are accessed at segment level. If the field is null, an appropriate value (typically one to three) is selected as default at the time the file is opened.

recordring - This field specifies the highest ring from which record-level access requests may be issued. If the field contains a null value, the default values depend on the access level selected and are shown below:

Record Level	- ring of user or ring of FAP (if a FAP is defined)
Block Level	- ring one (record-level access is prohibited)
Segment Level	- ring one (record-level access is prohibited)

blockring - This field specifies the highest ring from which block-level access requests may be issued. If the field contains a null value, the default values depend on the access level selected and are shown below:

Record Level	- ring of system record management procedures
Block Level	- ring of user or ring of FAP (if a FAP is defined)
Segment Level	- ring one (block-level access is prohibited)

NOTE: If neither recordring nor blockring values are specified, the resulting defaults cause the file to be processed at the same access level each time it attached or opened. By specifying ring values, the owner can permit the file to be processed at different access levels. The owner must be aware, however, that while processing a file at block level that was created at record level is a potentially useful feature, the converse is not allowed. Note further that setting all ring values to one effectively prohibits all access to the file, even by the system.

writering - This field defines the highest write ring for files which allow segment-level access. If null, the system-supplied default depends upon the Access Level selected for the file and is shown below.

Record Level	- ring of system record management procedures
Block Level	- ring of user or ring of FAP (if FAP is defined)
Segment Level	- ring of user or ring of FAP (if FAP is defined)

readring - This field defines the highest read ring for files which allow segment-level access. If null, the system-supplied defaults are identical to those described for the writering field.

NOTE: Since the writering/readring fields map directly to the R1/R2 fields defined by IPL hardware and since they also establish an execute bracket for segments which have execution enabled, the values supplied for the two fields may not be lower than the ring from which the FM#CREATE_FILE or FM#REDEFINE_FILE requests are issued if the file has Library organization. Since files having Library organization are the only files which can have "execute" usage selected, this prevents a user from directly creating an executable file which can execute at a ring lower than his current ring of execution.

recformat - This field defines the record format for this file and may contain one of five values:

F - fixed length records, no control fields are present.

D - variable length records, each record is prefixed with a four-byte control field which specifies (in decimal) the length of the record including the control fields.

S - variable length spanned records, each record is prefixed with a five byte control field consisting of a one-byte spanning indicator and a four-byte length field.

Y - variable length records, each record is prefixed with a four-byte control field containing a number of binary sub-fields.

U - undefined records, no control fields are appended and each record is equivalent to a block.

Record formats F, D, S and U conform to ANSI tape interchange standards. Y format records are used by the system and are the default selection if the field is null.

recsize - This field defines the maximum record size (in bytes) for the file. If null, the default selection depends on the file organization and hardware type but generally is equal to the block size less control field overhead.

keyloc - This field is applicable to files with Indexed organization. It contains the location, relative to the start of a data record, of the first byte of the primary key (the first byte of a data record is numbered 1) and has no default value.

keysize - This field is applicable only to files with Indexed organization. It contains the length, in bytes, of the primary key and has no default value.

loading - This field is applicable only to files with Indexed organization. It specifies the loading percentage to be used when initially writing the file (a loading percentage of N means to fill all data blocks to approximately N percent of their maximum capacity) and has a default value of one hundred.

compression - This field has meaning only for sequentially organized files containing Y-format records which permit data compression. If the field contains the value Y, records are compressed by the standard system algorithm before placing them in block buffers. If the field contains the value N or null, records are not compressed.

NOTE: This facility is intended primarily for preparation of files which are to be transmitted over low-speed serial communication lines. If subsequent design information on related components of the IPL hardware/software system negates the anticipated use of this facility, it will be dropped.

spanopt - This field has meaning only for sequentially organized files containing Y-format records. If the field contains the value Y, records are permitted to span blocks. If the field contains the value N or null, records are not permitted to span blocks.

NOTE: The major benefit of spanned records is that full utilization of space within a block is assumed. The major disadvantage is that records may not be updated if they are permitted to span blocks.

filesize - This field specifies the maximum size allowable for the file. It is expressed in units of blocks and has meaning only for mass storage files. If the field is null, an installation-defined percentage is applied to the initial file size in order to allow subsequent expansion.

expansion - This field specifies the expansion increment to be applied by the system whenever an implicit expansion of the file is required. It is expressed in units of blocks and has meaning only for mass storage files. If the field is null, an installation-defined default value is selected.

vord - This field specifies the ordinal (within the volume set identified by the "location" field of This File Control Block) of the volume on which to restrict allocation whenever an implicit expansion of the file is required. The field has meaning only for mass storage files. If the field contains all ones or is null, all volumes of the volume set are considered candidates for allocation.

verify - This field specifies whether write verification is to be used when the file is processed and has meaning only for mass storage files. A value of Y selects write verification a value of N or null suppresses write verification.

userdata - This field contains a pointer to a user-supplied 32-byte data area which will be saved with the cataloged description of the file. The field has meaning only for permanent mass storage files and has no default value. By supplying this pointer before the initial catalog entry is created for the file and supplying it before subsequently attaching the file, a facility roughly equivalent to user tape labels is provided.

fapdata - this field contains a pointer to a FAP-supplied data area which will be saved with the cataloged description of the file. The field has meaning only for permanent mass storage files and has no default value. By supplying this pointer before the initial catalog entry is created for the file and supplying it before subsequently attaching the file, the File Access Procedure is able to describe additional information it requires to process the file.

sequence - This field defines the sequence number of this file on a multifile tape volume set and has no default value. File sequence numbers are generated when files are initially written and are used for positioning when existing files are opened.

section - This field specifies the current section number of a multivolume tape file and has no default value. The field is

set to the value 0001 on the first section of the file and is incremented by one on each subsequent volume of the file.

hdroption - This field specifies whether the standard four-byte block header is present and has meaning only for tape files processed through record-level access procedures. If the file contains the value N, block headers are suppressed on output and assumed not to be present on input. If the field contains the value Y or null, the standard block header for sequential fields (containing a two-byte binary block length and a two-byte binary block sequence number) is generated on output and checked on input.

bsnoption - This field specifies whether the block sequence number contained in the standard block header is present and has meaning only for tape files processed through record-level access procedures. If the field contains the value N, the block sequence number in the block header is set to zero on output and is not checked on input. If the field contains the value Y or null, block sequence numbers are generated on output and checked on input:

NOTE: Proper use of hdroption and bsnootion fields allows a number of "industry standard" tape formats to be processed directly by IPL record-level access procedures. Non-standard codes and non-standard record dellmiting conventions must, of course, be handled in other ways.

linesize - This field specifies the maximum number of characters allowed on a single line and has meaning only for output to devices for which a "line" is defined. If null, an appropriate default value is selected for the current hardware type.

pagesize - This field specifies the maximum number of lines on a single page and has meaning only for output to devices for which a "page" is defined. If null, an appropriate default value is selected for the current hardware type.

formcntl - This field specifies the number of the forms control matrix to use when printing this file and has meaning only for printers which operate with a selectable forms control matrix. If the field contains all ones or is null, the "standard" forms control matrix is selected.

papercntl - This field specifies the number of the paper motion matrix to use when printing this file and has meaning only for printers which operate with a selectable paper motion

7.0 APPENDIX A - FIELDS OF A FILE CONTROL BLOCK

matrix. If the field contains all ones or is null, the "standard" paper motion matrix is selected.

linedelim - This field specifies the character to be interpreted as a line delimiter and has meaning only for input from keyboard devices. If null, an appropriate delimiter is selected for the current hardware type.

backspace - This field specifies the character to be used for character deletion and has meaning only for input from teletype-like devices which transmit a character at a time. If null, an appropriate character is selected for the current hardware type.

msgdelim - This field specifies the character to be interpreted as end-of-message and has meaning only for terminal devices which operate in binary delimited mode. No default value is defined.

8.0 APPENDIX B - VALID CONCURRENT FILE SHARING SELECTIONS

Current Attach Selection	Valid Open Selections (if ACL permits)	Valid Attach Selections In Other Jobs					
		E	PI	PU	PX	UI	UU
F	I, O, E, U, X	-	-	-	-	-	-
PI	I	-	OK	-	-	OK	-
PU	I, U	-	-	-	-	OK	-
PX	X	-	-	-	OK	-	-
UI	I	-	OK	OK	-	OK	OK
UU	I, U	-	-	-	-	OK	OK

9.0 APPENDIX C - RECORD MANAGEMENT REQUEST SUMMARY

	VALID REQUEST PARAMETERS														VALID REQUESTS BY OPEN USAGE					VALID FILE ORG/HOW TYPE			
	INPUT									I/O		OUTPUT								IND	REL	SEQUENTIAL	
	od	ba	bl	ka	kl	fai	co	lo	wo	dd	bf	fao	tc	rf	I	O	E	U	X	MASS STORAGE			OTHER
RM#GET	*	*	*	—	—	—	—	*	*	Φ	Φ	*	*	*	*	—	—	*	—	*	*	*	*
RM#GETP	*	*	*	—	—	—	—	—	—	—	—	—	*	*	*	—	—	*	—	—	—	*	*
RM#GETK	*	*	*	*	—	—	—	*	*	Φ	Φ	*	*	*	*	—	—	*	—	*	*	—	—
RM#GETD	*	*	*	—	—	*	—	*	*	Φ	Φ	*	*	*	*	—	—	*	—	*	*	*	—
RM#PUT	*	*	*	—	—	—	—	*	—	I	I	*	*	—	—	*	*	*	—	*	*	*	*
RM#PUTP	*	*	*	—	—	—	—	—	—	I	I	*	*	—	—	*	*	*	—	—	—	*	*
RM#PUTK	*	*	*	*	—	—	—	*	—	I	I	*	*	—	—	*	—	*	—	*	*	—	—
RM#REPLACE	*	*	*	—	—	—	—	—	—	—	—	*	*	—	—	—	—	*	—	*	*	*	—
RM#REPLACEK	*	*	*	*	—	—	—	—	—	—	—	*	*	—	—	—	—	*	—	*	*	—	—
RM#REPLACED	*	*	*	—	—	*	—	—	—	—	—	*	*	—	—	—	—	*	—	*	*	*	—
RM#DELETE	*	—	—	—	—	—	—	—	—	—	—	*	—	—	—	—	—	*	—	*	*	Y	—
RM#DELETEK	*	—	—	*	—	—	—	—	—	—	—	*	—	—	—	—	—	*	—	*	*	—	—
RM#DELETED	*	—	—	—	—	*	—	—	—	—	—	*	—	—	—	—	—	*	—	*	*	Y	—
RM#FINDF	*	—	—	—	—	—	—	—	—	—	—	*	—	—	*	—	—	*	—	*	*	*	*
RM#FINDP	*	—	—	—	—	—	—	—	—	—	—	*	—	—	*	—	—	*	—	—	—	*	*
RM#FINDK	*	—	—	*	*	—	*	—	—	—	—	*	—	—	*	—	—	*	—	*	*	—	—
RM#FINDD	*	—	—	—	—	*	—	—	—	—	—	*	—	—	*	—	—	*	—	*	*	*	—
RM#LOCK	*	—	—	—	—	*	—	*	*	—	—	—	—	—	*	—	—	*	—	*	*	*	—
RM#UNLOCK	*	—	—	—	—	*	—	—	—	—	—	—	—	—	*	—	—	*	—	*	*	*	—

LEGEND:
* - THE PARAMETER HAS MEANING TO THE REQUEST
— - THE PARAMETER IS IGNORED BY THE REQUEST
I - THE PARAMETER IS INPUT TO THE REQUEST
Φ - THE PARAMETER IS OUTPUT FROM THE REQUEST

* - VALID
— - INVALID

* - VALID REQUEST
— - INVALID REQUEST
Y - VALID REQUEST FOR Y-FORMAT RECORDS

PARAMETER DESCRIPTIONS					
CATEGORY	ABBREV	PARAMETER NAME	TYPE	SIZE	NOTES
INPUT PARAMETERS	od	opendesc	pointer	6	pointer to Open Descriptor
	ba	bufadr	pointer	6	pointer to buffer
	bl	buflength	integer	2	length of buffer (in bytes)
	ka	Keyadr	pointer	6	pointer to record key
	kl	keylength	integer	1	length of record Key (in bytes)
	fai	fileadrin	integer	8	input file address
	co	compareopt	char	1	=/>/≥ (EQ/GT/GE)
	lo	lockopt	char	1	E/S (Exclusive/Shared)
	wo	waitopt	char	1	W/R (Wait/Reject)
I/O PARAMETERS	dd	delim	char	1	data delimiting character
	bf	binflag	boolean	1/8	binary flag (0=coded, 1=binary)
OUTPUT PARAMETERS	fao	fileadrout	integer	8	output file address
	tc	count	integer	2	transfer count (in bytes)
	rf	recordflag	boolean	1/8	full/partial record (0=full record)

10.0 APPENDIX D - BLOCK MANAGEMENT REQUEST SUMMARY

	VALID REQUEST PARAMETERS												VALID REQUESTS BY OPEN USAGE					VALID REQ. BY HOW TYPE		
	INPUT										RESPONSE									
	rba	st	od	ecb	ba	bl	bn	f	lo	wo	tc	r	I	O	E	U	X	MS	MT	Que
BM#READ	*	*	*	*	*	*	–	–	–	–	*	*	*	–	–	*	–	*	*	*
BM#READD	*	*	*	*	*	*	*	–	–	–	*	*	*	–	–	*	–	*	–	–
BM#WRITE	*	*	*	*	*	*	–	–	–	–	*	*	–	*	*	*	–	*	*	*
BM#WRITED	*	*	*	*	*	*	*	–	–	–	*	*	–	*	–	*	–	*	–	–
BM#POINTF	*	*	*	*	–	–	–	–	–	–	–	*	*	–	–	*	–	*	*	–
BM#POINTL	*	*	*	*	–	–	–	–	–	–	–	*	*	–	–	*	–	*	*	–
BM#POINTP	*	*	*	*	–	–	–	–	–	–	–	*	*	–	–	*	–	*	*	–
BM#READSTATUS	*	*	*	*	*	*	–	–	–	–	*	*	*	*	*	*	–	*	*	*
BM#SELECT	*	*	*	*	*	*	*	*	–	–	*	*	*	*	*	*	–	*	*	*
BM#CANCEL	*	*	–	–	–	–	–	–	–	–	–	–	*	*	*	*	–	*	*	*
BM#LOCK	*	*	*	–	–	–	*	–	*	*	–	–	*	–	–	*	–	*	–	–
BM#UNLOCK	*	*	*	–	–	–	*	–	–	–	–	–	*	–	–	*	–	*	–	–

LEGEND:			
*	-THE PARAMETER HAS MEANING TO THE REQUEST	*	THE REQUEST IS VALID
–	-THE PARAMETER IS IGNORED BY THE REQUEST	–	THE REQUEST IS INVALID

PARAMETER DESCRIPTIONS					
CATEGORY	ABBREV.	PARAMETER NAME	TYPE	SIZE	NOTES
Input	rba	rba	pointer	6	pointer to Request Block
Output	st	status	record	8	request status
Input	od	opendesc	pointer	6	pointer to Open Descriptor
Input	ecb	ecbptr	pointer	6	pointer to Event Control Block
Input	ba	bufadr	pointer	6	pointer to buffer
Input	bl	buflength	integer	2	length of buffer (in bytes)
Input	bn	blocknum	integer	4	block number
Input	f	function	integer	1	device-dependent function
Input	lo	lockopt	char	1	E/S (Exclusive/Shared)
Input	wo	waitopt	char	1	W/R (Wait/Reject)
Output	tc	count	integer	2	transfer count (in bytes)
Output	r	response	record	8	status of operation

11.0 APPENDIX E - ERROR CODES

12.0 APPENDIX F - SCL VOLUME/FILE MANAGEMENT REQUESTS

To be supplied.

13.0 APPENDIX G - SWL REPPRESENTATION OF REQUEST BLOCKS

To be supplied

14.0 APPENDIX H - DATA MANAGEMENT REQUEST MACROS

To be supplied.