SPERRY UNIVAC

OWEN
TOWNSEND
UNIVAC VANCOUVER

no. CS -14 rev. no. 2 date: 4/9/80

OS/3 COMPONENT PRODUCT SOFTWARE DESCRIPTION

name: COMMON DATA INTERFACE (TO DATA MANAGEMENT)

component no.:

author: C. Gross/M. Goldberg

hardware systems: ☒ SUL ☐ 90/25 ☒ 90/30 ☒ 90/40

☐

ABSTRACT:

This document defines a Common Data Interface (CDI) that consists of a single uniform set of declarative and imperative directives to provide the necessary linkage between a calling program and the appropriate Data Management processor(s).

Significant modifications are indicated by a vertical bar in the right-hand margin.

APPROVALS:

section manager:	director
group manager:	
director:	

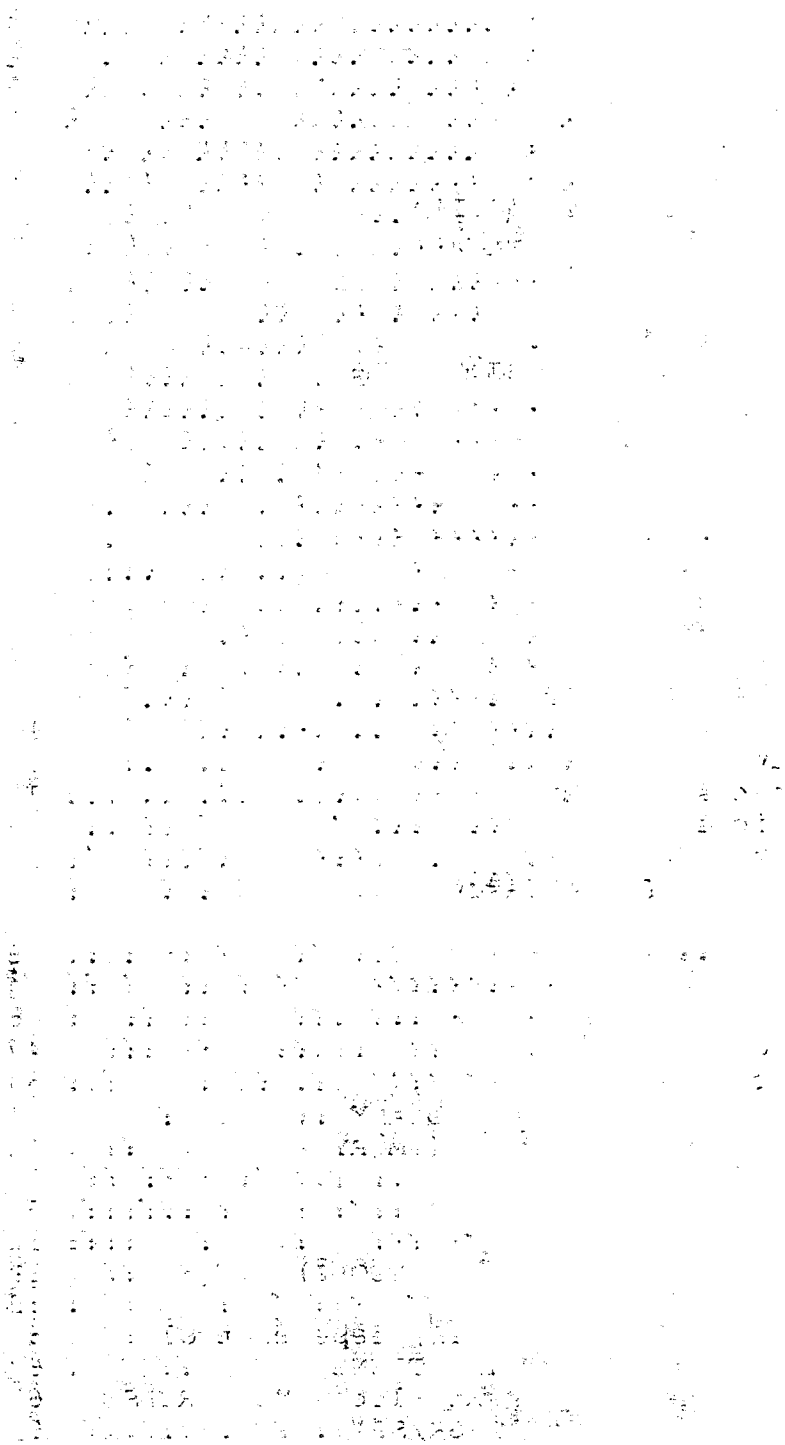


TABLE OF CONTENTS

TABLE OF CONTENTS (Continued) PAGE

1. INTRODUCTION

1.1 Scope

This document outlines the functionality of the Common Data Interface and the user interfaces that are required. It adheres to the specifications detailed in PD A-41058, SUL S-2 System.

1.2 Purpose

The common data interface offers a standardized procedure for passage of functional requests or data to separate processors within the Data Management system. While such a generalized approach to an interface could be adapted to other processes within a system or used to cross communication lines to non-native handlers, this document identifies only Data Management interfaces and potential CDI applications are left open-ended.

2. COMPONENT SUMMARY

2.1 CDI Strategy

Within the current OS/3 operating system, every Data Management access method defines its own declarative control structures (DTFs) and maintains separate and unique generators for each. Status posting and error presentations are defined by the requested processor; functional requests are identified by a set of imperatives which often relate only to an individual access method. With system supplied DSECTs, the user is very much aware of control structure organizations and is able to directly reference and modify these tables without verification by Data Management. Inadvertent modification of a DTF could directly effect the integrity of a file and/or the actions taken by its processor.

The CDI Data Management system consists of a single uniform interface which isolates the user from the format of these control structures; critical processor dependent control information is maintained outside of the user address space. The user is restricted from directly modifying these control structures and potentially distorting the actions of the Data Management system. Special imperatives provide the user the indirect means to reference and modify these controls but the system retains

control over which changes are permitted or rejected. A standardized set of declarative and imperative definitions provide the mechanism for logical level access to all devices, independent of device characteristics.

All CDI Data Management is offered as a set of system components utilized in a shared code environment, external to the user program. This prohibits the user from directly modifying Data Management code and thereby distorting any subsequent functional requests. All Data Management shared code will execute under key zero which is inaccessible to the user.

2.2 Data Management Controls

In the CDI Data Management system, device/processor dependent control structures are removed from the users address space. The structures, relocated to key zero memory, will not be directly accessible to the user thereby providing increased reliability. Access method declaratives (DTFs) are replaced by two other directives to provide communication with the Data Management system. One is a functional request mechanism (CDIB) independent of the logical processor which services the request; the other, a parameter passing mechanism (RIB) independent of the control structure to which it applies. These user specified declaratives, along with the Data Management generated control structures (FPBs and FMBs) will be discussed in the following subsections.

2.2.1 Common Data Interface Block (CDIB)

The Common Data Interface Block is a generalized functional passing mechanism. It provides a standard method for identifying a resource, indicating a desired function through interface controls and receipt of status. The CDIB is the users window to the Data Management system that enables logical level functionality which is both independent of device assignment or the access method required. The actual relationship between the logical and physical device is established during resource initialization (OPEN). The specific device dependent control structure will be generated in the system memory pool. The CDIB and Resource Information Block (RIB) together define the symbolic information that will affect that generation.

The CDIB is a user resident, encoded structure that may someday enable logical file processing through a non-native handler (distributed). The description of a CDIB, its fields and contents, is discussed in Section 4.1.1 of the User Interfaces and Appendix A.

2.2.2 Resource Information Block (RIB)

The Resource Information Block is an encoded parameter list used to pass symbolic information between the user and a processor. It is presented in conjunction with a CDIB for resource initialization (OPEN) or when modifying or interrogating device dependent control structures. The RIB generator produces a variable length encoded list based on the parameters specified; unnecessary parameters need not be specified nor is space allotted.

During OPEN, the RIB enables the user to specify parameters as he did with OS/3 DTF declaratives. However, with the exception of address type specifications, the SUL OPEN will assign default values to the fields of device dependent control structures based on the specific device assigned in job control. The user may elect to have the default specification or may override it by definition. A RIB used for the OPEN process need not be maintained for the course of the user program; rather, its space may be reused or overlayed as required.

On a post-OPEN basis, the user may wish to view the contents of the processor structures, i.e. fields, conditions, etc. This information was maintained in the OS/3 DTF and required little effort to reference the data. With CDI structures maintained in key zero memory, the user must identify a CDIB and RIB through the DMQRY imperative. Additionally, the user may wish to modify the contents of the processor dependent structures. A CDIB and RIB combination specified through the DMAPY imperative will provide that service.

The parameters that can be specified in the Resource Information Block are defined in Section 4.1.2 of the User Interfaces. They are presented in a device class order to provide familiarity with device requirements. Any specifications across device classes may be used; parameters that do not relate to the device that is assigned will be ignored.

2.2.3 File Processing Block (FPB)

The File Processing Block represents the external device/processor dependent control structure and is not unlike the OS/3 DTF. It is generated by the OPEN process; the type of access method structure being determined by the specific device assignment. It will be maintained by the Data Management system the system memory pool, covered by key zero, and may not be directly modified by the user program.

There will be an FPB structure for every access method. Direct/immediate reference to a specific FPB will be achieved by a directory index inserted in the CDIB at OPEN.

2.2.4 File Management Block (FMB)

The File Management Block is the central control mechanism for all physical disk file information. It, too, will be generated by the Data Management OPEN process and maintained in the system memory pool. There will be no facility to enable a user program to directly or indirectly modify this structure.

There will be one FMB for every active disk file in the system. It is used to maintain physical file extents, end of data identification and filelocking controls. Subsequent OPENs of an active disk file will result in the assignment of the same FMB. An FMB will remain active while at least one user has the file OPEN. FMB space may be reused for other file definitions when the previous file is no longer in use.

2.3 Data Management Design

For CDI Data Management, all processor dependent control structures, are removed from the users address space. Interfaces between the user and Data Management are made via the system SCALL SVC (an integral part of every DM imperative).

The SCALL SVC will enter the SVC decode portion of the supervisor. An acceleration is achieved by immediate recognition of the SCALL request; no priority scan will be envoked. Register enstack/destack services will replace user defined storage areas. Internal requests for other shared code elements will also have stacking provisions. Internal and external shared code requests will be made by element index; shared code exits will request destacking until the last causes a return to the user program.

All Data Management imperatives will request the Centralized File Manager. It is the focal point for control of the Data Management system; it is the beginning of every logical process. It converts CDIB references to the appropriate FPB and FMB structures, controls address resolutions and governs filelock granularity. The type of FPB control structure defines the specific shared code element to receive control.

All Data Management is defined as key zero shared code. This single form of Data Management eliminates the support of redundant processors (linked or assembled).

There are two general processor types: (1) System Access Technique (SAT) processors, and (2) Non-SAT processors. Device independence is primarily supported between processors of the same type (i.e. SAT or non-SAT). A single, non-SAT, processor is provided for each logical device (MIRAM, the only disk access method, provides sequential, random and index functionality).

In addition, a single SAT processor is provided for each logical device.

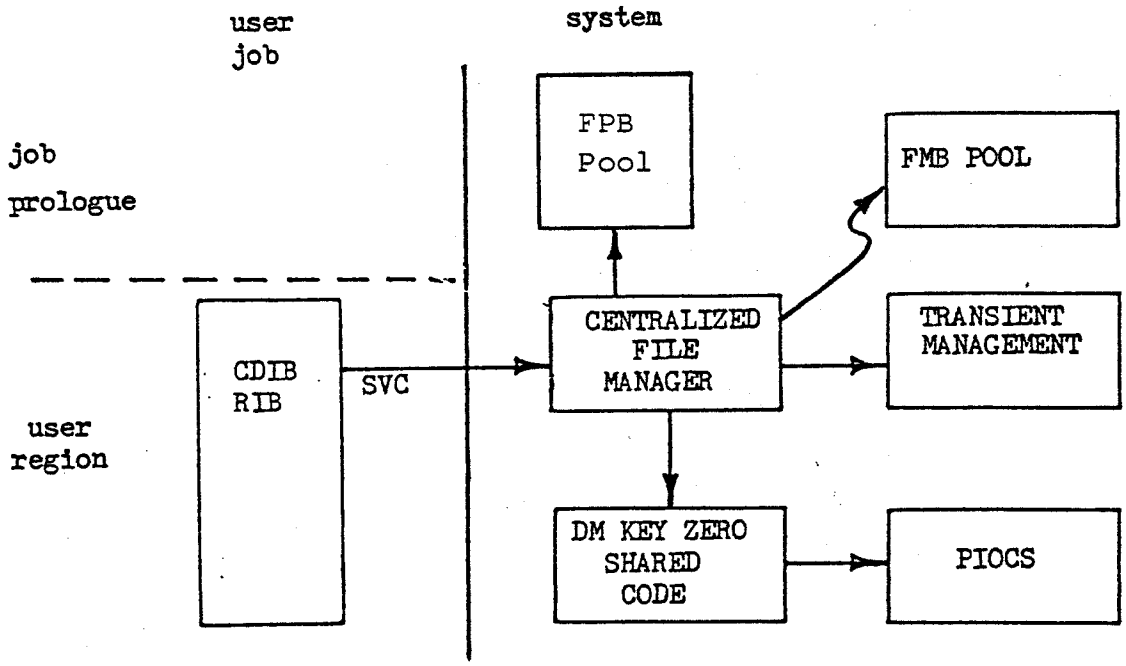


Figure 1: CDI Data Management Process

3. DESIGN TRADE-OFFS AND PRODUCT OBJECTIVES

Some of the major objectives of this new means of interfacing to Data Management are:

- o establishing common control structures to be used to interface to DM, regardless of the device being accessed;
- o providing a higher degree of device independence (than what was previously available) by delaying until file OPEN, the determination of what device type is being accessed and which access method should be used;
- o establishing a central point of control (CFM) which is necessary to direct control to the appropriate access method, perform address resolution, provide additional file lock facilities, provide distributed processing functionality, etc.

4. INTERFACE DESCRIPTION

4.1 User Interfaces

The interfaces between the user and the CDI Data Management system no longer reference a DTF structure. Rather, the file OPEN procedure develops the structural requirements from a combination of device assignment and characteristics, label information and user defined specifications. The structures themselves are not maintained in the user's address space which therefore insulates the user from the structures and vise versa. User access to them is via special interface controls: new declarative definitions and a consolidated set of imperatives that map to the functionality of all devices. This section formalizes these user interfaces.

4.1.1 Declarative Macros

4.1.1.1 CDIB Macro

The Common Data Interface Block declarative defines a functional request mechanism that identifies a resource (by filename) and reserves interface and completion data areas. A user may reference externally generated control structures only through a CDIB, it is the primary reference in all of the standard set of CDI imperatives.

A CDIB is a device independent structure that can be used to reference more than one resource (by changing the filename). So long as the CDIB identification, length and filename fields are intact, any CDIB can be used to reference any resource. It need not be the one with which the OPEN was requested nor must the same CDIB be used for CLOSE.

The format of a CDIB declarative is:

NAME	OPERATION	OPERAND
[cdibname]	CDIB	[FILENAME=filename]

cdibname: cannot exceed 7 characters.

FILENAME=filename: filename cannot exceed 8 characters. If specified, the CDIB will be expanded using the corresponding name in the filename field of the CDIB (i.e. DC CL8 'filename'); otherwise, the cdibname will be placed in the filename field (i.e. DC CL8 'cdibname'). If neither is specified, the user is responsible for moving the filename into the appropriate field.

A complete layout of the CDIB and a definition of the individual fields can be referenced in Appendix A. DSECT names and status conditions are also included.

4.1.1.2 RIB macro

4.1.1.2.1 RIB definition

The RIB declarative macro is the means by which data management users describe their logical file characteristics and file processing requirements. This description consists of specifying the appropriate keyword parameters on the RIB macro call line. The set of allowable keyword parameters encompasses device independent as well as device dependent file processing specifications for all the devices supported by data management. Processing specifications for several devices can be described within a single macro call.

The RIB declarative does not activate the Data Management control system, but merely generates a structure (Resource Information Block) within the users region. Information within the RIB structure is the encoded representation of the keywords cited on the macro call line. The size of the RIB structure varies depending upon the number of keywords specified.

4.1.1.2.2 RIB usage

The RIB must always be presented to the Data Management control system in combination with a CDIB which identifies a logical filename. Data Management is able to associate, through the job control device assignment set, the logical filename to a device type. A processor dependent control structure (FPB), located outside the users region, is required by the logical IOCS processor to control operations to the device.

The OPEN imperative generates this FPB and initializes it with default parameters based on the device assigned. If the user's logical file characteristics and file processing requirements coincide with the default specifications, then a RIB need not accompany a CDIB for the OPEN imperative. When a RIB is supplied, OPEN scans the RIB structure for all device dependent as well as device independent parameters and applies them to the FPB structure, tailoring it to represent the user's file processing needs.

Unlike the OPEN imperative, the RIB is required by the DMAPY and DMQRY imperatives. These imperatives permit the post-open altering/referencing of selected fields within the FPB structure. See appendix B.4 for the RIB keywords acted upon by the DMAPY and DMQRY imperatives.

4.1.1.2.3 Specifying RIB Parameters

There are two ways of specifying RIB keyword parameters - the direct and indirect method. With the direct method, the keyword data represents the actual specification. Some examples of direct parameter definitions are:

```
RIB BFSZ=256,IOA1=BUFF,WORK=YES
```

For the indirect method the keyword data on the macro call line cites the address of where the actual specification can be found. At the specified address the user is responsible for generating data (outside the RIB structure) that represents the desired specification. This user generated data must be compatible (in terms of content and length) to the data generated by the RIB macro if the keyword were specified by the direct method.

The indirect format is useful for users that determine their file characteristics/processing requirements at program execution, or for those users that change their processing requirements during program execution. Secondly, indirect specifications are required for those parameters the user wishes to query.

Some examples of indirect parameter definitions are:

```
RIB BFSZ=.DSBFSZ,IOA1=.ABUFF,WORK=.WRKSPEC
```

Notice that the indirect specifications require a period (.) prefix before the symbolic address to distinguish it from a direct specification.

Indirect and direct specifications can be incorporated within the RIB macro call. An example would be:

```
RIB BFSZ=256,IOA1=.ABUFF,WORK=.WRKSPEC
```

Although the indirect method offers greater flexibility, it does require knowledge of each specification format (value and length). Data management attempts to facilitate this through the RIBEQU proc which allows the user to generate this data symbolically. The RIBEQU proc is just a set of equates that symbolically define all RIB keywords and their associated specifications. To illustrate this, reference Appendix B.1. Example #1 lists 3 items: (1) five RIB equate keywords and their associated specifications extracted from RIBEQU, (2) a RIB macro call with direct specifications, and (3) the generated RIB structure.

Example #2 is the same RIB macro call but with indirect specifications and demonstrates how the user would generate indirect data symbolically using equates from the RIBEQU proc.

Some general comments regarding the RIB structures outlined in examples #1 and #2 of appendix B.1:

- The RIB structure is a series of define constants (DCs) whose lengths and values are derived from the RIB equates. These equates can be displayed by calling the RIBEQU proc before the first RIB macro call in your assembly.

- The RIB structure is demarcated by a RIB start and RIB end identifiers (2 bytes each).

- Entries within the RIB structure, between the start and end sentinels, are composed of a keyword identifier (2 bytes) and a data portion whose length varies depending upon the keyword.

- Indirect data lengths can be generated by using the length attribute of the keyword identifier symbol defined in the RIB equates.

- Indirect data values can be generated by an address constant of one of the specification symbols defined in the RIB equates for that particular keyword.

4.1.1.2.4 Miscellaneous Items

Several miscellaneous items regarding the RIB are listed below:

- Address specifications. Data management expects all address specification to reside in a three byte field. The RIB macro turns out three byte ADCONs for direct specifications such as IOA1, IOA2, etc. The user must also generate three byte ADCONs for indirect address specifications. If Data Management detects three bytes of zeroes for any address specification, it assumes the user is negating this specification. For example, a program with an indirect specification for LABADDR may determine that on a particular OPEN to a tape file, user labels are not to be processed. By zeroing out the three bytes of indirect data (address specification), the user would nullify the LABADDR specification. Therefore, the user should not use a symbolic address (for address specifications) that generates a relative displacement of zero.

- Base/Displacement. A form of relocation has been incorporated for addresses generated within the RIB. This facility is known as base/displacement and is useful for Base Ø and Key Ø shared code modules that must present a RIB with varying specifications. See appendix B.2 for details.

- Dynamic RIB generations. Some applications need to generate RIBs without calling the RIB macro (although not recommended). Appendix B.3 identifies data management aids for this very special application.

4.1.1.2.5 Access Method RIB Parameters

This section will describe the allowable RIB parameters for the following devices:

- DISK
- TAPE
- PRINTER
- CARD(READER/PUNCH)

Each device can be accessed via SAT or NON-SAT interfaces.

CDI interfaces to the following devices are described in the corresponding documents:

DISKETTE: CS-36

WORKSTATION: CS-39

4.1.1.2.5.1 SAT Interfaces

To be supplied.

4.1.1.2.5.2 Non-SAT Interfaces

4.1.1.2.5.2.1 Disk (MIRAM) Specifications

NAME	OPERATION	OPERAND
[ribname]	RIB	[, ACCESS= {EXC / SRDO / EXCR / SRD / SADD}]
		[, AUTOIO= {NO / YES}]
		[, BFSZ= n]
		[, INDA=symbol]
		[, INDOUT= {NO / YES}]
		[, INDS= {Ø / n}]
		[, IOA1=symbol]
		[, IOA2=symbol]
		[, IORG= {NO / (r)}]
		[, KARG=symbol]
		[, KEY$_n$=([size] , [loc] , [{NDUP / DUP}], [{NCHG / CHG}])]

(continued)

NAME	OPERATION	OPERAND
[ribname]	RIB	[, MODE= {SEQ / RAN / RANH}]
		[, NUMREC=symbol]
		[, NWAIT= {NO / YES}]
		[, OPTN= {NO / YES}]
		[, OUTF= {NO / YES}]

(continued)

INTENTIONALLY LEFT BLANK

Disk (MIRAM) Specifications

NAME	OPERATION	OPERAND
		[, PROC= {UNK / KEY / INDO}]
		[, RCB= {NO / YES}]
		[, RCFM= {FIX / VAR}]
		[, RCSZ= n]
		[, RETR= {INF / REP / MOD}]
		[, SCSZ= {n}]
		[, SKAD=symbol]
		[, TRUNC= {NO / YES}]
		[, VMNT={NO / ONE}]
		[, VRFY= {NO / YES}]
		[, WKFM={NO / VAR / VARI}]
		[, WORK= {NO / YES}]

Parameters:

ACCESS= { EXC | SRDO | EXCR | SRD | SADD }

Specifies the file share environment. The environment dictates whether multiple "logical access paths" (i.e. LAPs) can access the physical file at the same time, and the type of processing (i.e. read and/or write) that each LAP can perform. When a user attempts to open a file (i.e. create a LAP), if there are no other LAPs for that physical file, then the user's ACCESS specification will dictate the file share environment. If there already are LAPs for the file, then the user's ACCESS specification is compared with the file share environment that has been previously established. If they are compatible, then the file is opened and a new LAP is created. If they are not compatible, then the task is waited until the ACCESS specification and the environment are compatible. (See the NWAIT RIB parameter if a wait condition is not desirable.)

EXC - EXCLUSIVE environment. Only 1 LAP is permitted. This LAP wants read/write use of the file, and no other LAPs are permissible. No shareability exists in this environment. (This is the default.)

SRDO - READ ONLY environment. This LAP wants only read use of the file and is willing to share the file with other LAPs that want only read use. Other participants in this environment must specify ACCESS=SRDO.

EXCR - SINGLE WRITER environment. This LAP wants read/write use of the file and is willing to share the file with other LAPs that want only read use. Other participants in this environment must specify ACCESS=SRD.

SRD - SINGLE WRITER environment. This LAP wants only read use of the file and is willing to share the file with one user who wants read/write use and/or multiple users who want only read use. Other participants in this environment must specify ACCESS=EXCR or SRD.

SADD - MULTIPLE WRITER environment. This LAP wants read/write use of the file and is willing to share the file with other LAPS that want read/write use. Other participants in this environment must specify ACCESS=SADD.

Since multiple LAPs can be writing to the file in this environment, D.M. must perform some special processing. Certain function requests (i.e. output, input with the intent to update, etc.) will result in a set of blocks (i.e. sectors) being locked to the job. Dead locks are avoided by preventing a job from accumulating more than one set of block locks. (See Appendix E for additional information.)

AUTOIO= {NO / YES}

AUTOIO=YES will result in the following functions being performed for DISK(MIRAM) files:

- o force the write of the data buffer (i.e. no buffered data left in Core).
- o force the wait of data buffer write
- o force the read of data buffer (i.e., at start of imperative process, buffer is assumed to contain "garbage")
- o force the write of index buffer (i.e., no buffered data left in core; index buffer writes are always waited)
- o force the read of index buffer (i.e., at start of imperative process, buffer is assumed to contain "garbage")
- o no "look-ahead"performed (i.e. I/O overlap) for unkeyed sequential input

(This specification is only permitted under single buffering. It is provided for the user who wants to: (1) dynamically change buffer addresses while the file is open; and (2) use a workarea but does not want any unwaited I/O issued. Since all I/O's are forced, performance will be degraded. The minimum allowable data buffer size should be specified to avoid wasted buffer space and I/O's of unnecessary sectors.)

BFSZ= n

specifies the size of the data buffer in the file, where n is the size, in bytes. The size must be at least "sector size" as well as a multiple of "sector size."

The algorithm for determining the "minimum allowable value" in sectors is as follows:

$$\frac{L}{S} = N + R$$

where: L ≡ slot size
S ≡ sector size
N ≡ number of "full" sectors per slot (i.e. quotient)
R ≡ remainder

The minimum number of sectors per buffer is:

N; if R=Ø
N+1; if R divides evenly into S (i.e. no remainder)
N+2; otherwise

The default sector size is 256 (see SCSZ parameter for additional information on varying the sector size).

(NOTE: For files with RCB and fixed length records, slot size=record size + 1; otherwise slot size=record size.)

(See Appendix C for additional information on OPEN label checking and default values.)

INDA=symbol

specifies the symbolic address of where index blocks are processed during keyed operations. It must be half-word aligned. The length of the area is specified by the INDS parameter. This area must immediately precede the primary I/O buffer (IOA1).

In order for index operations (keyed or index only) to be permitted, all of the index related keywords must be specified: INDA, INDS, KARG and KEY1. If any are missing, it will be assumed that index operations were not intended to be employed. (See KEY_n keyword for the single exception to this rule.)

INTENTIONALLY

LEFT

BLANK

INDOUT= { NO / YES }

This parameter is used to indicate the user's intention with respect to writing index-only "records" in the file.

NO - index-only output is inhibited.

YES - index-only output is permitted.

This parameter only pertains to files which do not have the record control byte (see RCB parameter). For files that have the RCB, index-only output is always permitted (i.e. INDOUT=YES is assumed).

(See Appendix C for additional information on OPEN label checking and default values.)

INTENTIONALLY

LEFT

BLANK

INDS= { Ø / n }

specifies the length of the index area (INDA), where n is the length in bytes. If specified, it must be a multiple of 256. It is required for all index operations.

(See Appendix C for additional information on OPEN label checking and default values.)

IOA1=symbol

specifies the location of the I/O area, where symbol is the location. Must be half-word aligned. The length of the area is specified by the BFSZ parameter. It must immediately follow the index buffer (INDA) if specified. It must also immediately precede the secondary I/O buffer (IOA2) if specified, unless index operations are not to be performed.

A file which can perform index operations must have all buffers contiguous.

(See Appendix D for information on dynamic buffering.)

IOA2=symbol

specifies the location of an additional I/O area. It must be half-word aligned and of the same size as the primary I/O area (IOA1). If index operations are to be performed, this buffer must immediately follow IOA1. Use of a secondary buffer is only permitted when performing sequential output (keyed or unkeyed) or unkeyed sequential input operations.

IORG= { NO / (r) }

specifies the general register to be used to point to the current record in the I/O area when the user is not referencing records in the workarea. Registers 2 through 12 are available. Specifying IORG=(r) automatically sets the WORK=NO specification (i.e. WORK=NO does not have to be specified, it is assumed).
(See WORK parameter for additional information.)

KARG=symbol

specifies the field in the user's program where he will place the keys to effect retrieval of records. The length of the KARG area is equal to the largest key length plus 3 (6 minimum). Required for all index operations.

INTENTIONALLY

LEFT

BLANK

KEY$_n$=([size] , [loc] [, { NDUP / DUP }] [, { NCHG / CHG }])

specifies one of up to five keys for an indexed file ($1 \leq n \leq 5$). Permitted size is 1 through 80 bytes. The loc parameter specifies the number of bytes preceding the key. (If loc is omitted, Ø is assumed; do not omit for variable records.) DUP specifies that duplicate keys are allowed (NDUP indicates that they are not allowed and is the default). CHG specifies that keys can change during update (NCHG indicates it cannot change and is the default).

(See Appendix C for additional information on OPEN label checking and default values.)

MODE= { SEQ / RAN / RANH }

This specification conditions the external control structure should the corresponding positional parameter be defaulted on the DMINP or DMOUT macros:

SEQ- Sequential (default)
RAN - random
RANH - random with hold

NUMREC= symbol

Specifies the location of the 3 byte field which will receive the record count (i.e. high record number in the file, including deleted records). This parameter is used if the user wants to interrogate the external control structure to determine the record count. It is only used when performing a query operation; therefore, it must always be specified "indirectly."

INTENTIONALLY

LEFT

BLANK

NWAIT=	NO YES	Specifies whether or not the task is to be waited when a file share environment incompatibility occurs (See the ACCESS RIB parameter for information on file share environment): NO- task is waited until the environments are compatible (default). YES- task is not waited. Error status is reported on the open call (error code = X'88') indicating a file share environment incompatibility.
OPTN =	NO YES	See the description of the optional file indicator, CD$IOPT (in the interface status section of Appendix A.1).
OUTF=	NO YES	Specifies that after every output to the file, the last record retrieved will be read back into the user's buffer.

INTENTIONALLY
LEFT BLANK

PROC= { UNK / KEY / INDO }

This specification conditions the external control structure should the corresponding positional parameter be defaulted on the DMINP, DMOUT or DMSEL (Record) macros:

UNK- Unkeyed (default)
KEY - Keyed
INDO - index only

RCB= { NO / YES }

This parameter is used to indicate the presence or absence of the record control byte; the RCB is necessary in order to perform logical record deletion (see DMDEL macro).

NO - no RCB is desired.

YES - RCB is desired.

(See Appendix C for additional information on OPEN label checking and default values.)

The location of the RCB varies depending on the record format (see the RCFM parameter for additional record format information):

fixed - RCB is appended to the front of each record. Workareas do not have to be large enough to hold the RCB. On input requests, the record which is presented to the user, via I/O register or workarea, will not contain the RCB.

variable - The third byte of the RDW (record descriptor word) is used as the RCB.

In additional to record deletion capabilities, the following functionality is provided if there is an RCB: (1) Sequential read will skip over any deleted records; (2) random read of a deleted record will result in a no-find error condition; (3) random output beyond the last record in the file will result in the "gap" being filled with deleted records; (4) random output within the file limits will receive an error condition if a valid (e.g. not deleted) record is at the specified location; (5) "mixed" files can be created. A "mixed" file is one which contains keyed and unkeyed records.

INTENTIONALLY
LEFT BLANK

RCFM={ FIX / VAR }

This parameter is used to indicate the record format.

FIX - fixed length records.

VAR - Variable length records. Variable length records are supported within a fixed size slot, where slot size = record size. The first 4 bytes of the slot is the record descriptor word (RDW), and the first 2 bytes of the RDW hold the record size (user supplied on output).

For device independence, the FIXUNB, FIXBLK or UNDEF specifications will be treated as FIX, VARUNB or VARBLK will be treated as VAR.)

(See Appendix C for additional information on OPEN label checking and default values.)

INTENTIONALLY
LEFT BLANK

RCSZ= n

Specifies the length in bytes of each record. For variable length records, this should reflect the slot size and should include the 4 byte overhead. The presence (or absence) of the RCB does not affect the RCSZ specification. (See Appendix C for additional information on OPEN label checking and default values.)

RETR= {INF / MOD / REP}

This specification conditions the external control structure should the corresponding positional parameter be defaulted on the DMINP macro:

INF - informational purposes (default)
MOD - modification
REP - replacement

SCSZ= n

Specifies the physical sector size, which for selector channel devices (i.e. 8414, 8425, 8430, 8433) can vary from the default value of 256. Only values greater than (or equal to) 256 will be permitted. (On sectorized devices, this specification is ignored and the default value, 256, will always be used.) The purpose for increasing the sector size would be: (1) to increase the effective capacity of the disk (i.e. less hardware overhead for sectorization), or (2) to force an integer number of records to fit within a buffer, which will improve performance for sequential operations. (See Appendix C for additional information on OPEN label checking and default values.)

INTENTIONALLY

LEFT

BLANK

SKAD = symbol

Specifies the location in the user's program into which he loads the relative disk address for use in processing files by relative record number. The SKAD field is a 4-byte fullword aligned field. The first record is relative record one.

This parameter must be specified if the following operations are to be performed (i.e. optional otherwise):

- o unkeyed random input
- o unkeyed random output
- o index only random output
- o index only input (random or sequential)
- o unkeyed select record

If this parameter is specified and the function request was successful, the following information is presented back to the user in the 4 byte field:

- o number of records in the file + 1 (OPEN process only)
- o relative record number of the record in question (DMINP, DMOUT and DMSEL (RECORD) processes only)

TRUNC={ NO / YES }

Specifies the manner in which input truncation will be reported. (Input truncation occurs when the actual size of the record in question exceeds that which the program requested. For disk, tape and card files, it can only occur in conjunction with the WKFM=VARI specification.)

NO - truncation is ignored (default). Successful status is reported and the program is not aware that truncation occurred.

YES - truncation is reported as exception status. The device independent indicator, CD$TRUNC (bit 5 of filename C byte 3), will be set on to indicate exception status due to an "input truncation" condition. (See the INTERFACE STATUS section of Appendix A.1 for additional information.)

VMNT= { NO / ONE }

Specifies that the file is to be processed with only one volume online at anytime. A file which is created in this manner must be processed likewise. Non-keyed random or keyed random output operations will not be permitted. (The default specifies that the file is always to be processed with all volumes online.) See Appendix C for additional information on OPEN label checking and default values.)

VRFY= { NO / YES }

specifies that Data Management is to check parity of output records after they have been written to disk. Use of this parameter increases execution time for output functions by one revolution period per block. If bad parity is detected, Data Management will return Output Parity Check status in the CDIB and return to the user inline. If omitted, no output parity verification will be done.

INTENTIONALLY

LEFT

BLANK

WKFM= { NO / VAR / VARI }

Specifies that the workarea (which is to be used by Data Management) is in variable format (i.e. 4 byte record descriptor word followed by data). The workarea format may be different from the record format (RCFM specification). Workarea processing must be employed. All record sizes are passed in the first 2 bytes of the record descriptor word, RDW, and include the 4 byte RDW.

NO - Specifies that the workarea format is identical to the record format. This is the default.

VAR - Specifies that the workarea format is variable. For output operations, the user must specify (in the RDW) the effective size of the record. For input operations, Data Management will specify (in the RDW) the effective size of the record. Since the entire record will always be moved into the workarea on input operations, the workarea must be large enough to hold the largest record which could potentially be read.

VARI - Specifies that the workarea format is variable and will be used in the same manner as VAR for output operations. For input operations, the user must specify the maximum number of bytes to be transferred to the workarea. If the specified size is less than the actual record size, the record will be truncated. (See the TRUNC parameter for additional information on the status that is reported.) If the specified size is greater than the actual record size, the value in the RDW will be changed to reflect the actual size of the record moved to the workarea (and successful status is reported).

WORK= { YES / NO }

YES (the default) specifies that the user will be processing records in a work area and not in the I/O area. The address of the work area is specified with each issue of the appropriate imperative. If both IORG=(r) and WORK=YES are explicitly specified in the RIB, work area processing is assumed and the IORG specification is ignored. When work area processing is being used, the work area and the I/O area should not be the same buffer.

Special considerations for disk (MIRAM) files only:

- work area processing must be used for all output (DMOUT), keyed update (DMUPD) or keyed delete (DMDEL) functions.
- If both IORG=(r) and WORK=YES are explicitly specified in the RIB, the IORG specification is ignored (as previously discussed), but with one exception: on an input function (DMINP) for replacement (RETR=REP) the user's I/O register will be pointed to the record in the I/O buffer.
- The workarea and the I/O area can be the same buffer under the following conditions:
  - unkeyed output (DMOUT) operations only
  - RCB=NO
  - AUTOIO=YES
  - record size = buffer size

4.1.1.2.5.2.2 Tape Specifications

NAME	OPERATION	OPERAND
[ribname]	RIB	[,ASCII=YES]
		[,AUTOIO={NO / YES}]
		[,BFSZ= n]
		[,BKNO={NO / YES}]
		[,BUFOFF={Ø / n}]
		[,CKPTREC={NO / YES}]
		[,CLRW={UNLOAD / YES / NO}]
		[,ERROPT={NO / SKIP / IGNORE}]
		[,FILABL={NO / NSTD / STD}]
		[,IOA1=symbol]
		[,IOA2=symbol]
		[,IORG={NO / (r)}]

(Continued)

Tape Specifications

NAME	OPERATION	OPERAND
		[,LENCHK={NO / YES}]
		[,OPRW={NO / YES}]
		[,OPTN={NO / YES}]
		[,RCFM={FIXUNB / FIXBLK / UNDEF / VARUNB / VARBLK}]
		[,RCSZ={n / (r)}]
		[,READ={FORWARD / BACK}]
		[,TPMARK={NO / YES}]
		[,TRANS={NO / ASCII}]

(continued)

Tape Specifications

NAME	OPERATION	OPERAND
		[, TRUNC={NO / YES}]
		[, TYPEFLE={INPUT / OUTPUT}]
		[, ULABEL={NO / YES}]
		[, VARBLD=(r)]
		[, WKFM={NO / VAR / VARI}]
		[, WORK={NO / YES}]

INTENTIONALLY

LEFT

BLANK

Parameters:

ASCII=YES

If ASCII processing is desired for tape files, this specification must be made. The RIB proc will generate the appropriate "A-con" and EXTRN to force auto-inclusion of the ASCII translate table module (DT$ETA;512 bytes). The address of the table will only be used if TRANS=ASCII is specified. (i.e. for TRANS=NO, the table address will be ignored). (Specifying TRANS=ASCII and ASCII=YES is identical to specifying ASCII=YES in the DTFMT call under DTF interfaces.)

AUTOIO={NO / YES}

AUTOIO=YES will result in the following functions being performed for tape files:

- o force the wait of buffer write
- o no "look-ahead" performed (i.e. I/O overlap) on input

(This specification is only permitted under single buffering with unblocked records. It is provided for the user who wants to: (1) dynamically change buffer address while the file is open; and, 2) use a workarea but does not want any unwaited I/O issued. Use of this parameter will degrade performance.)

BFSZ= n

specifies the length, in bytes, of the I/O area.

(See Appendix C for additional information on OPEN label checking and default values.

BKNO={NO / YES}

specifies that block numbers are to be created during output operations and that the sequence of numbers be checked during input. If YES, it requires that a 4-byte storage area, full-word aligned, precede each buffer used.

BUFOFF={Ø / n}

specifies the length of a block prefix in bytes (for ASCII files only), where Ø≤n≤99.

CKPTREC={NO / YES}

specifies that checkpoint blocks are to be bypassed on input files. If omitted, the default of NO processes checkpoint records as data. The CKPTREC parameter is ignored for an output file.

CLRW={UNLOAD / YES / NO}

specifies the disposition of tape volumes after CLOSE:

UNLOAD - tape is rewound and unloaded. This is the default. (This is identical to defaulting the CLRW specification in the DTFMT call under DTF interfaces.)

YES - tape is rewound and not unloaded. (Identical to CLRW=RWD under DTF interfaces)

NO - tape is not rewound. (Identical to CLRW=NORWD under DTF interfaces.)

(See OPRW specification for additional information)

ERROPT={NO / SKIP / IGNORE}

specifies the options available when a unique error occurs while processing the tape. The SKIP option will cause Data Management to bypass an input record containing a parity error. The record is not made available for processing. When IGNORE is specified, the parity error is ignored and the record can be processed as though no error has been detected. The NO option (default) will cause DM to return with an unsuccessful indication when a parity error is detected.

FILABL={NO / NSTD / STD}

specifies whether the file contains standard or nonstandard labels or is unlabeled. NO (the default option) indicates that the first volume and all other volumes are unlabeled. If NSTD, the file does not contain labels that conform to SUL-OS/3 standards. User processing of nonstandard labels is defined by the LABADDR parameter. The STD option identifies the file containing standard labels that conform to standard conventions.

IOAl=symbol

specifies the location of the I/O area. If BKNO=YES then 4-bytes must be reserved immediately preceding the I/O area. Both the 4-byte field and the I/O area must be full-word aligned. When BKNO is not specified, the I/O area must only be half-word aligned.

(See Appendix D for information on dynamic buffering.)

IOA2=symbol — specifies the address of an optional secondary I/O area for the file, subject to the same requirements (e.g. size, alignment, etc) as IOA1.

IORG={NO / (r)} — (See IORG description under disk (MIRAM) specifications)

LENCHK={NO / YES} — YES specifies, for ASCII input files, that data management is to check the block length specified in the block prefix of variable length records against their physical record length.

OPRW={YES / NO} — specifies whether reels of the file are to be rewound before labels are checked during file OPEN:

YES - tape is rewound. This is the default. (This is identical to defaulting the OPRW specification in the DTFMT call under DTF interfaces.)

NO - tape is not rewound. (Identical to OPRW=NORWD under DTF interfaces.)

(Under DTF interfaces there is the REWIND parameter which is redundant. The equivalent functionality of specifying REWIND=NORWD is to specify OPRW=NO and CLRW=NO. The equivalent functionality of specifying REWIND=UNLOAD is to specify OPRW=YES and CLRW=UNLOAD, both of which are the defaults.)

OPTN={NO / YES} — See the description of the optional file indicator, CD$IOPT (in the interface status section of Appendix A.1).

RCFM= { FIXUNB / FIXBLK / UNDEF / VARUNB / VARBLK }

FIXUNB - fixed length unblocked records

FIXBLK - fixed length block records

UNDEF - undefined

VARUNB - variable length unblocked records

VARBLK - variable length blocked records

(For device independence, the FIX specification will be treated as FIXUNB; VAR will be treated as VARUNB.) (See Appendix C for additional information on OPEN label checking and default values.)

RCSZ= { n / (r) }

specifies the number of bytes in each record. The n option is used to define fixed-length record sizes, the (r) option must be used when reading or writing undefined records; r is the general register to contain the block length of each undefined record.

(See Appendix C for additional information on OPEN label checking and default values.)

READ= { FORWARD / BACK }

specifies the direction the tape file is to be read. FORWARD reading is the default; BACK indicates that the file is to be read backward. Multivolume files cannot be read backward as the volume serial number is not present in the volume trailer labels.

TPMARK= { NO / YES }

specifies, for output files with non-standard labels or no labels that Data Management is to either write or not write a tape mark to separate labels from data. If NO, it becomes the user's responsibility to distinguish between labels and data.

TRANS= { NO / ASCII }

NO - specifies that no translation is to be performed (default).

ASCII - Specifies that the file is to be processed as an ASCII file. (see the ASCII keyword parameter for additional information)

(TRANS=YES is not supported by tape and will be ignored.)

TRUNC= { NO / YES } (See TRUNC description under disk (MIRAM) specifications.)

TYPEFLE = { INPUT / OUTPUT } INPUT specifies that the file is an input file, to be read. No output function can then be issued to the file.

OUTPUT specifies the file is an output file, to be written. No read can be issued in that state.

If not specified, the default will be INPUT, unless file extension is indicated, in which case, OUTPUT will be set.

INTENTIONALLY

LEFT

BLANK

ULABEL= { NO / YES }

Specifies whether user header labels (UHLs) and user trailer labels (UTLs) are to be processed. On a standard labeled file, there is a maximum of 8 UHLs and 8 UTLs and they must follow the standard 80-byte format. On a non-standard labeled file, there are no limitations with respect to number or content.

NO - no user label processing is to be performed (default). On an output file, no user labels can be written. On an input file, any user labels will be bypassed.

YES - user label processing is to be performed. During file or volume open and close processing, when label processing is necessary, the appropriate function request (OPEN, CLOSE, DMFEV, DMINP or DMOUT) is interrupted and exception status is returned. The device dependent indicator, CT$LABEL, will be set on to indicate exception status due to a "label processing required" condition. One of the following alphabetic characters will have been placed in the device dependent field, CT$LCODE, to indicate the corresponding process being performed:

- o C'O' at file or volume open
- o C'V' at volume close
- o C'F' at file close

A device dependent indicator, CT$INPUT, will be set to indicate whether input or output label processing is required ("on" for input; "off" for output).

The user is responsible for transferring control to a user label processing routine where the DMLAB macro must be issued to process the labels. (No other macro can be issued against the file until label processing is terminated.) The DMLAB (IO) macro must be issued to read or write successive labels. The DMLAB (END) macro must be issued to terminate label processing and complete the original function request.

If the original function was an OPEN or CLOSE, when the DMLAB (END) macro is issued, the open or close process will be completed and the appropriate status will be returned.

(continued)

If the original function was a DMFEV, DMINP or DMOUT, label processing is necessary during the close of the current volume and the open of the next volume. When the DMLAB (END) macro is issued, the close process is completed, the volumes are swapped and then the open process is started. If label processing is necessary during the open of the new volume, the status that is returned on the DMLAB (END) macro is exception status due to a "label processing required" condition (exactly the same as that which was previously reported on the original function request). The user should transfer control to the beginning of the label processing routine and the DMLAB (IO) macro should be used during this volume open procedure. When the DMLAB (END) macro is again issued, the open process is completed as well as the original function request and the status that is returned is the status of the DMFEV, DMINP or DMOUT function.

Note: When the DMFEV macro is issued to an input file, label processing at volume close is bypassed.

(See the INTERFACE STATUS section of Appendix A.1 for additional information on exception status.)

VARBLD= (r)

specifies the general register that Data Management loads with the number of bytes of residual space left in the current I/O area. It is only used for output files with variable length blocked records processed in an I/O area and cannot be used when records are processed in a work area.

WKFM= { NO / VAR / VARI }

(See WKFM description under disk (MIRAM) specifications.)

WORK= { NO / YES }

(See WORK description under disk (MIRAM) specifications.)

4.1.1.2.5.2.3 Printer Specifications

NAME	OPERATION	OPERAND
[ribname]	RIB	[,BFSZ= n]
		[,CONTROL={YES / CTL}]
		[,IOA1=symbol]
		[,IOA2=symbol]
		[,IORG={NO / (r)}]
		[,OPTN={NO / YES}]
		[,PRAD={1 / n}]
		[,PRINTOV={SKIP / REPORT / YES}]
		[,RCFM={FIXUNB / VARUNB / UNDEF}]
		[,RCSZ= (r)]
		[,TRUNC={NO / YES}]
		[,UCS={OFF / ON}]
		[,WKFM={NO / VAR / VAR1}]
		[,WORK={NO / YES}]

Parameters:

BFSZ= n

specifies the length, in bytes, of the I/O area. If omitted, the block size will be established so that the number of characters printed will be 120. By using the RCFM keyword (and accounting for an 8 byte overhead for variable length records) and by using the CONTROL keyword (and accounting for a 1 byte overhead for a control character,) the buffer size will be set to 120, 121, 128, or 129. If a value is specified which is greater than what is required, it will be rounded down to the appropriate value (depending on the particular device and the RCFM and CONTROL specifications). The two overheads are not printed, but the buffer must be big enough to hold them (if necessary).

Note: To print an odd number of characters, allocate I/O buffers one byte larger than the number of characters to be printed.

CONTROL= {YES / CTL}

YES- specifies that spacing or skipping of lines is to be controlled via the DMCTL (SK or SP) imperatives. (This is identical to specifying CONTROL=YES in the DTFPR call under DTF interfaces.)

CTL - specifies that a device independent control character will be used with data records. (This is identical to specifying CTLCHR=DI in the DTFPR call under DTF interfaces.)

IOA1=symbol

specifies the location of the I/O area. It must be aligned so the first byte of data (the character to be printed in column 1) is on a half-word boundary. It should be allocated in even numbers of bytes (excluding the control character).

(See Appendix D for information on dynamic buffering.)

IOA2=symbol

specifies the address of an optional secondary I/O area for the file, subject to the same requirements (e.g. size, alignment, etc.) as IOA1.

IORG= { NO / (r) } (See IORG description under disk (MIRAM) specifications.)

OPTN= { NO / YES } See the description of the optional file indicator, CD$IOPT (in the interface status section of Appendix A.1).

PRAD= { 1 / n } specifies a standard forms advance from 1 to 15 lines. The form advance takes place after the line is printed. A default of 1 is assumed unless CONTROL=CTL is specified, then the control character determines the line advancement.

PRINTOV= { SKIP / REPORT / YES }

SKIP - Upon detecting a forms overflow condition, DM will automatically skip to the home paper position. No indication will be passed back to the user that an overflow condition was encountered. This is the default. (This is identical to specifying PRINTOV=SKIP in the DTFPR call under DTF interfaces.)

REPORT - Upon detecting a forms overflow condition, DM will inform the user by reporting exception status (see the INTERFACE STATUS section of Appendix A.1 for additional information). The original imperative will have been performed correctly. (This is the substitute for specifying PRINTOV=symbol in the DTFPR call under DTF interfaces.)

YES - specifies that the DMCTL (PRTOV) macro will be issued by the user in order to control forms overflow detection and actions. No indication will be passed back to the user that a forms overflow condition was encountered. (This is identical to specifying PRINTOV=YES in the DTFPR call under DTF interfaces.)

RCFM= { FIXUNB / VARUNB / UNDEF }

FIXUNB - Fixed length unblocked records (default)

VARUNB - Variable length unblocked records

UNDEF - Undefined records

(For device independence, the FIX or FIXBLK specifications will be treated as FIXUNB; VAR or VARBLK will be treated as VARUNB.)

RCSZ= (r)

Specifies the number (2 through 12) of the register that holds the size of the output record. (Required for undefined record format)

TRUNC= { NO / YES }

(See TRUNC description under disk (MIRAM) specifications.)

UCS= { OFF / ON }

specifies whether character mismatches are to be ignored or not. Mismatches occur whenever the printer attempts to print a bit configuration which is not present in the printer's load code buffer. The default (OFF) option indicates that mismatches are to be ignored. The non-printable character is replaced by a blank. If ON, the operator will be informed of mismatches via a message from PIOCS.

WKFM= { NO / VAR / VARI }

(See WKFM description under disk (MIRAM) specifications.)

WORK= { NO / YES }

(See WORK description under disk (MIRAM) specifications.)

4.1.1.2.5.2.4 CARD (READER and PUNCH) Specifications

NAME	OPERATION	OPERAND
[ribname e]	RIB	[,AUE={NO / YES}]
		[,BFSZ= n]
		[,CDMODE={STD / CC / BINARY}]
		[,CONTROL= YES]
		[,IOA1= symbol]
		[,IOA2= symbol]
		[,IORG={NO / (r)}]
		[,ITBL= symbol]
		[,OPTN={NO / YES}]
		[,ORLP={NO / YES}]
		[,OTBL= sYmbol]
		[,OUBLKSZ= n]
		[,RCFM={FIXUNB / VARUNB / UNDEF}]

(Continued)

CARD (READER and PUNCH) Specifications

NAME	OPERATION	OPERAND
		[,RCSZ= (r)]
		[,STUB={NO / 51 / 66}]
		[,TRANS={NO / YES / ASCII}]
		[, TRUNC={NO / YES}]
		[,TYPEFLE={INOUT / OUTPUT / INPUT}]
		[,WKFM= {NO / VAR / VARI}]
		[,WORK= {YES / NO}]

PARAMETERS:

AUE={NO / YES}

Specifies for Data Management to inhibit error processing when validity checks are detected on nonbinary input files. Validity errors will cause a PIOCS message indicating the problem. The card containing the error will be the last card in the stacker. By operator option, the card can be reread. If the default or NO option is used, validity check error will cause the unique unit error indicator to be set in the CDIB.

BFSZ= n

Specifies the length, in bytes, of the I/O area. If omitted, the block size is determined from the RCFM keyword specification. For fixed length records, BFSZ will default to 80; for variable length records BFSZ will default to 84 (i.e., the buffer includes the block and record headers, but only the record header is punched).

For variable length records, the BFSZ value (whether defaulted or user specified) reflects the maximum size for records.

If a value is specified which is greater than what is required, it will be rounded down to the appropriate value (depending on the particular device and the RCFM, STUB and MODE specifications).

Note: To read or punch an odd number of columns, allocate I/O buffers one byte larger than the BFSZ value.

CDMODE={STD / CC / BINARY}

specifies the input/output mode of the file. (This parameter is equivalent to the MODE keyword parameter in the DTFCD proc under DTF interfaces.)

STD- specified for cards which are to be punched or read in EBCDIC. This is the default. This form must also be used for ASCII translation (see TRANS keyword parameter).

CC - specified for cards read or punched in compressed code.

BINARY - specifies column binary mode. A buffer size and I/O area of 160 bytes is required for one 80-column card.

CONTROL= YES

There is no choice involved, as YES is the default and the only choice. This specifies that DMCTL, SS macros will be issued to control stacker selection. (This is identical to specifying CONTROL= YES in the DTFCD call under DTF interfaces. The functionality that was provided by specifying CRDERR=RETRY under DTF interfaces, is not available.)

IOA1=symbol

specifies the location of the I/O area where symbol is the location. The I/O area should be defined as an even number of bytes; the first character of the area must be half-word aligned.. The length of the area is specified by the BFSZ parameter.

(See Appendix D for information on dynamic buffering.)

IOA2=symbol

specifies the address of an optional secondary I/O area for the file, subject to the same requirements (e.g. size, alignment, etc.) as IOA1.

IORG={NO / (r)}

(See IORG description under disk (MIRAM) specifications.)

ITBL=symbol — specifies the location of the 256-byte user defined translation table to be used for input translation. (See TRANS keyword for additional information)

OPTN= { NO / YES } — (See the description of the optional file indicator, CD$IOPT (in the interface status section of Appendix A.1).

ORLP= { NO / YES } — YES specifies that a combined file is to be processed in an overlap mode.

OTBL=symbol — specifies the location of the 256-byte user defined translation table to be used for output translation. (See TRANS keyword for additional information.)

OUBLKSZ= n — specifies the length of the secondary I/O buffer for a combined file. If omitted the BFSZ value is used.

RCFM= { FIXUNB / VARUNB / UNDEF }

FIXUNB - fixed length unblocked records (default). Must be used for input or combined files.

VARUNB - variable length unblocked records. Only permitted for output files.

UNDEF - Undefined records. Only permitted for OUTPUT files.

(For device independence, the FIXBLK specification will be treated as 'FIXUNB; VARBLK will be treated as VARUNB.)

RCSZ= (r) — specifies the number (2 through 12) of the register that holds the size of the output record. (Required for undefined record format.)

STUB= { NO / 51 / 66 } — Specifies that the stub card read feature applies to cards with 51 or 66 columns. The default (NO) is that standard 80-column cards are assumed.

TRANS={NO / YES / ASCII}

specifies the translation that is to be performed:

NO - no translation is to be performed. (This is the default.)

YES - translation is to be performed per user specification. The ITBL or OTBL parameters (as appropriate) must be specified. When specified, the CDMODE specification will be ignored. (This is identical to specifying MODE=TRANS in the DTFCD call under DTF interfaces)

ASCII - specifies that the system supplied ASCII translate table is to be used. (This is identical to specifying ASCII=YES in the DTFCD call under DTF interfaces.)

TRUNC={NO / YES}

(See TRUNC description under disk (MIRAM) specifications.)

TYPEFLE={INPUT / OUTPUT / INOUT}

This parameter only applies to the punch device that has the read features; for readers, or punches without read features, this parameter is ignored and the external control structure is automatically established to reflect an input or output file.

For this punch device with read features, combined file processing (INOUT) is the default. If the device is intended to be used as strictly a reader or punch, then INPUT or OUTPUT should be specified.

WKFM={NO / VAR / VARI}

(See WKFM description under disk (MIRAM) specifications.)

WORK={NO / YES}

(See WORK description under disk (MIRAM) specifications.)

(Blank Page)

4.1.2 Imperative Macros

If a file is "opened using CDI interfaces" (i.e. by specifying a CDIB via symbolic address or register notation on the OPEN call), then:

- The file must be "accessed using CDI interfaces" (i.e. the imperative macros that have the DM prefix must be used); and
- The file must be "closed using CDI interfaces" (i.e. by specifying a CDIB via symbolic address or register notation on the CLOSE call).

The OPEN and CLOSE macros are used for both CDI and DTF interfaces to data management. The processes "look at" the control structure (i.e. CDIB or DTF) which is specified and react accordingly.

When a file is opened, an "access path" is established to the file via the external control structures (e.g. FPB, FMB, etc.). It is not Data Management's intention that multiple tasks within a job share the same access path. D.M. is not going to require that the path be used by only one task, but it is the user's responsibility to guarantee that multiple tasks do not attempt to use the same path at the same time. An error will be reported when this condition is detected.

INTENTIONALLY

LEFT

BLANK

Register conventions:

All CDI D. M. processes (i.e. open, close, input, output, etc.) expect the following registers to be established as follows:

RØ: RIB address (for OPEN with a RIB, DMAPY, DMQRY)
value of zero (for OPEN without a RIB)
workarea address (for imperatives that employ workareas and if workarea processing is invoked)
Miscellaneous (certain forms of the DMCTL and DMSEL macros require that RØ contain "pass information")

R1: CDIB address

(NOTE: If symbolic notation is used to specify the CDIB, RIB or workarea, then the expansion of the imperative macro will load RØ and/or R1 with the appropriate address.)

Upon receiving control back from a CDI imperative macro all registers are returned unchanged. (Note: the expansion of the macro may modify RØ and/or R1 as described above.) The only exception is that upon receiving control back from the OPEN and CLOSE processes, R14 will be destroyed.

4.1.2.1 Opening a File (OPEN)

The OPEN macro is the means by which the user initializes file resources. Only a single file can be defined. The open process will generate a processor dependent control structure(FPB) outside of the user region for the file being opened. The content of the FPB will be determined by both device characteristics of the device assigned and any exceptions specified by the user. The CDIB is the generalized request mechanism and defines the filename; the RIB is an optional list of parameters to effect the initialization. If the RIB is omitted, all specifications will be defaulted.

FORMAT:

NAME	OPERATION	OPERAND
[label]	OPEN	{cdibname / (1) / 1} [, {(ribname) / (Ø) / Ø}]

Positional Parameter 1:

Specifies the CDIB (via symbolic address or register notation) that contains the name(that was assigned through job control) of the file being opened.

Positional Parameter 2: (optional)

Specifies the RIB (via symbolic address or register notation). If register notation is used, a value of zero in register Ø will indicate that there is not a RIB to be employed (therefore a RIB cannot reside at relative address X'ØØØØØØ').

Notes:

- If symbolic notation is used for both parameters, the comma is optional.

(Blank page)

4.1.2.2 Closing a File (CLOSE)

The CLOSE macro affects the termination of file processing. For devices that require label services, they will be created or updated in accordance with the requirements of the file. Once closed, a file cannot be accessed until it is reopened with another OPEN imperative. Errors detected by CLOSE processing will be returned in the interface status field of the CDIB.

FORMAT:

NAME	OPERATION	OPERAND
[label]	CLOSE	{cdib name / (1) / 1}

The Operand specifies the CDIB (via symbolic address or register notation) that contains the name (that was assigned through job control) of the file being closed. No RIB specification is required during the CLOSE process or allowed on the macro.

Multiple files should not be specified on a single CLOSE call because it is cumbersome to check for errors. A series of CLOSE calls should be issued with an error check after each one.

4.1.2.3 Requesting a Record (DMINP)

The DMINP macro makes a record available for processing. It is a generalized input imperative that can be used to request records from all Data Management processors (printer excluded). However, there are parameters that relate specifically to DISK (MIRAM).

FORMAT:

NAME	OPERATION	OPERAND
[label]	DMINP	{cdibname / (1) / 1} [, {workarea / (Ø) / Ø}] [,{INF / MOD / REP}] [,{UNK / KEY / INDO}] [,{SEQ / RAN / RANH}]

Positional parameter 1:

Specifies the CDIB (via symbolic address or register notation) that contains the name (that was assigned through job control) of the file being accessed.

Positional parameter 2:

Specifies the workarea (via symbolic address or register notation) that will receive the record.

The following parameters and notes relate specifically to DISK (MIRAM). These parameters will be ignored when accessing other file types. If specified, they will temporarily override the default specification that was established in the external control structure at file OPEN; otherwise the defaults will be employed.

Positional parameter 3:

INF - retrieve for informational purpose only (update or delete requests will not be permitted).

MOD - retrieve for modification (update or delete requests will be permitted).

REP - retrieve for replacement(update or delete requests will be permitted). The record will not be moved into the workarea as it is assumed that a replacement record already resides in the workarea.

Positional parameter 4:

UNK - unkeyed retrieval
KEY - keyed retrieval
INDO - retrieval of index entry information.

Positional parameter 5:

SEQ - Sequential retrieval based on current sequential position (keyed sequential for KEY or INDO retrieval; next highest record number for UNK retrieval). Current sequential position is modified.

RAN - random retrieval based on user specified argument (in KARG for KEY or INDO retrieval; in SKAD for UNK retrieval) Current sequential position will be modified.

RANH - Same as RAN specification except that the current sequential position will be held (not modified).

NOTES: (For disk MIRAM) For successful operations the following information is available:

- o An indicator as to whether or not the acquired record is keyed.

(Continued)

- An indicator as to whether or not the next sequential record (in the current key of reference) has a duplicate key (to the acquired record).
- The relative record number (of the record in question) will be placed in the SKAD field (if one is specified).
- The key will be placed in the KARG field (for keyed records or index only entries).

(See page A10 for details on the above mentioned indicators that can be set.)

DMINP and DMSEL (RECORD) are the only macros that can affect the current sequential position.

4.1.2.4 Outputting a Record (DMOUT)

The DMOUT macro provides for placement of a new record in a file. It is a generalized imperative that can be used to present records to all Data Management processors. However, there are parameters that relate specifically to the DISK (MIRAM).

Format:

NAME	OPERATION	OPERAND
[label]	DMOUT	{cdibname / (1) / 1}[,[{workarea / (Ø) / Ø}]] [,{UNK / KEY / INDC}][,{SEQ / RAN / RANH}]

Positional Parameter 1: (Same as DMINP macro)

Positional Parameter 2:

Specifies the workarea (via symbolic address or register notation) that contains the record to be output.

The following parameters and notes relate specifically to DISK (MIRAM). These parameters will be ignored when accessing other file types. If specified, they will temporarily override the default specification that was established in the external control structure at file OPEN; otherwise the defaults will be employed.

Positional parameter 3:

UNK - unkeyed output.

KEY - Keyed output. Record is to be indexed according the key(s)of the file specification. An index entry (key and pointer) will be added to the index for each key in the file.

INDO - index only output. A single index entry will be added to the file in the current key of reference. The key must be supplied in the KARG area. The SEQ or RAN specification determines the relative record number to be used.

Positional parameter 4:

SEQ - record is to be placed at end of file (at the next available record position).

RAN - record is to be placed in a relative slot according to the record number given in the SKAD field.

RANH - (Same as RAN specification).

NOTES: (For disk MIRAM)

The following information is presented to the user upon completion of the function:

- An indicator will show if legal key duplication has occurred and on which keys.
- An indicator will show if illegal key duplication has occurred and on which keys.
- For successful functions, the relative record number (of the record in question) will be placed in the SKAD field (if one is specified).

(See page A10 for details on the above mentioned indicators that can be set.)

4.1.2.5 Control Structure Modification (DMAPY)

The DMAPY macro enables the user to permanently alter fields in an external processor dependent control structure (FPB). The modification can be performed immediately or for special applications, it can be performed repeatedly on every I/O imperative. (If the repeat facility is used, when an I/O imperative is issued, first the apply operation is performed and then the I/O function is performed.)

Each access method has a list of allowable apply parameters. (See Appendix B.4) Any parameters in the RIB that are not allowable for the particular access method will be ignored.

Format:

NAME	OPERATION	OPERAND
[label]	DMAPY	[cdibname / (1) / 1] , [ribname / (Ø) / Ø] , [[IMMED / REPEAT / END]]

Positional parameter 1: (Same as DMINP)

Positional parameter 2:

specifies the RIB (via symbolic address or register notation) that contains the list of logical parameters which will be used to permanently modify the control structure. (This parameter must be specified unless positional parameter 3 is end.)

Positional parameter 3:

REPEAT - The control structure will not be modified immediately upon execution of the imperative, but it will be on all subsequent I/O imperatives. Execution of this form of the imperative consists solely of saving the address of the RIB in the FPB and returning to the caller; no apply operation is performed. On all subsequent I/O imperatives, first the apply operation is performed (resulting in a permanent change to the control structure) and then the I/O function is performed.

Use of this form will override a previously issued DMAPY REPEAT imperative to the same file (i.e. the RIB address in the FPB will be changed).

END - This option will deactivate an outstanding DMAPY REPEAT function. Positional parameter 2 will be ignored and therefore need not be specified.

IMMED - The default is that the control structure will be modified immediately upon execution of the imperative. All changes made to the control structure are permanent. (This form of the imperative is <u>not</u> considered to be an I/O imperative, and therefore <u>no</u> repeat apply or query operations will be performed in conjunction with the execution of this "immediate" form of the imperative.)

4.1.2.6 Control Structure Interrogation (DMQRY)

The DMQRY macro provides the user with a facility for requesting the contents of a field (or fields) in an external processor dependent control structure (FPB). This interrogation can be performed immediately or for special applications, can be performed repeatedly on every I/O imperative. (If the repeat facility is used, when an I/O imperative is issued, first the I/O function is performed and then the query operation is performed.)

Each access method has a list of allowable query parameters. (See Appendix B.4). Any parameters in the RIB that are not allowable for the particular access method will be ignored. There is an additional requirement for the DMQRY macro: the parameters must have been specified using the indirect (dynamic) specification (i.e. IOA1=.TAG1). Direct specifications will be ignored even if the parameter is allowable for the particular access method.

Format:

NAME	OPERATION	OPERAND
[label]	DMQRY	{cdibname / (1) / 1} , {ribname / (Ø) / Ø} , [[{IMMED / REPEAT / END}]]

Positional parameter 1: (same as DMINP macro)

Positional parameter 2:

specifies the RIB (via symbolic address or register notation) that contains the list of logical parameters which will be used to interrogate the control structure. (This parameter must be specified unless positional parameter 3 is END.)

Positional parameter 3:

REPEAT - The control structure will not be interrogated immediately upon execution of the imperative, but it will be on all subsequent I/O imperatives.

Execution of this form of the imperative consists solely of saving the address of the RIB in the FPB and returning to the caller; no query operation is performed. On all subsequent I/O imperatives, first the I/O function is performed and then the query operation is performed.

Use of this form will override a previously issued DMQRY REPEAT imperative to the same file (i.e. the RIB address in the FPB will be changed).

END - This option will deactivate an outstanding DMQRY REPEAT function. Positional parameter 2 will be ignored and therefore need not be specified.

IMMED - The default is that the control structure will be interrogated immediately upon execution of the imperative. (This form of the imperative is not considered to be an I/O imperative, and therefore no repeat apply or query operations will be performed in conjunction with the execution of this "immediate" form of the imperative.)

4.1.2.7 Updating a Record (DMUPD)

The DMUPD macro causes the most recently retrieved record from a disk (MIRAM) file to be updated.

NAME	OPERATION	OPERAND
[label]	DMUPD	{cdibname / (1) / 1} ,[{workarea / (ø) / ø}]

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2:

specifies the workarea (via symbolic address or register notation) that contains the updated record. Workarea processing must be used if the original record is keyed.

NOTES: The following information is presented to the user upon completion of the function:

- an indicator will show if legal key duplication has occurred, and on which keys.
- an indicator will show if illegal key duplication has occurred, and on which key. (Only the first key in error is indicated).
- an indicator will show if an illegal key change has occurred, and on which key(only the first key in error is indicated).

(See page A10 for details on the above mentioned indicators that can be set.)

4.1.2.8 Deleting a Record (DMDEL)

The DMDEL macro causes the most recently retrieved record from a Disk (MIRAM) file to be deleted. It marks the subject record as void, and voids any index entries pointing to the record. The macro can only be issued against a file that has an RCB.

Format:

NAME	OPERATION	OPERAND
[label]	DMDEL	{ cdibname / (1) / 1 }

Positional parameter 1: (Same as DMINP macro)

4.1.2.9 Erasing a file (DMERS)

The DMERS macro is used to erase part of a Disk (MIRAM) file (which contains only unkeyed records) starting from a given relative record number.

Format:

NAME	OPERATION	OPERAND
[label]	DMERS	{cdibname / (1) / 1} , PART

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2:

PART causes all records, beginning with a specified record number (in SKAD), to be discarded. The record, which corresponds to the record number in the seek address field, and all records whose record numbers are greater in value, will be discarded. This macro can only be issued against a file which contains unkeyed records (i.e., no keyed records or index-only entries). (The RCB is not required to perform this function.)

4.1.2.1Ø Selecting a Record Search (DMSEL, RECORD)

The DMSEL RECORD macro prepares for making disk (MIRAM) file records available in sequential order by key or by record number. It can also be used to change the key of reference.

Format:

NAME	OPERATION	OPERAND
[label]	DMSEL	{cdibname / (1) / 1} ,RECORD [, {EQ / GT / GE / BOF / EOF}] [, {PKEY / KREF}] [, {KEY / INDO / UNK}]

Positional parameter 1: (Same as DMINP)

Positional parameter 2:

RECORD - identifies the record selection feature.

Positional parameter 3: (required unless KREF is specified in parameter 4, in which case parameter 3 is optional)

EQ - establish a sequential position at the record whose key (for KEY or INDO processing) or record number (for UNK processing) is equal to the specified argument.

GT - establish a sequential position at the record whose key (for KEY or INDO processing) or record number (for UNK processing) is greater than the specified argument.

(Continued)

GE - establish a sequential position at the record whose key (for KEY or INDO processing) or record number (for UNK processing) is greater than or equal to the specified argument.

BOF - establish a sequential position at the beginning of the file (at the record with the lowest record number for UNK processing or at the record with lowest key value for KEY or INDO processing).

EOF - establish a sequential position at the end of the file. Following this function with a sequential input function will result in an end-of-file condition.

Positional parameter 4: (only supported for KEY or INDO processing)

PKEY - selection is based on the n leading bytes of the specified argument ($n \leq$ key size). Register Ø must be preset with the value of n. If not specified, the full key will be used.

KREF - specified to change the "key of reference." Register Ø must be preset with the value n which corresponds to the KEYn RIB parameter ($1 \leq n \leq 5$ and n must not exceed the number of keys in the file). This specification can be used in conjunction with parameter 3, if first, a key of reference change and then a sequential positioning is desired.

Positional parameter 5: If specified, this parameter will temporarily override the default specification that was established in the external control structure at file OPEN; otherwise, the default will be employed.

UNK - positioning will be based on a relative record number which must be presented in the SKAD field (except for BOF and EOF).

KEY positioning will be based on a key value which must be presented in the KARG area (except for BOF and EOF). The current key of reference is used.

INDO - (Same as the KEY specification)

NOTES:

The following information is presented to the user upon completion of the function:

- If unsuccessful, error status will reflect a no-find condition.
- For successful functions, the relative record number (of the record pointed to) will be placed in the SKAD field (if one is specified), unless parameter 3 is EOF.
- For successful functions using KEY or INDO processing, the key (of the record pointed to) will be placed in the KARG field (unless parameter 3 is EOF)

When a file is opened, the sequential position (both keyed and unkeyed) is automatically established at the beginning of the file. Therefore it is not necessary to issue a "select beginning of file" macro (i.e. DMSEL 1, RECORD,BOF).

4.1.2.11 Terminating a Volume (DMFEV)

The DMFEV macro provides the capability to terminate processing on the current volume of the file for tape or disk (MIRAM) processed with only one volume online at any time. If issued against a file with all volumes mounted, the macro is ignored.

Format:

NAME	OPERATION	OPERAND
[label]	DMFEV	{cdibname / (1) / 1}

Positional parameter 1: (same as DMINP macro)

When issued, the current volume is closed and a mount message is issued requesting that the next volume of the file be mounted. The new volume of the file is opened for processing; subsequent macros will continue processing on the new volume.

4.1.2.12 Controlling Tape Functions (DMCTL, tape control)

The control macro is available to control the function of tape files.

Format:

NAME	OPERATION	OPERAND
[label]	DMCTL	{ cdibname / (1) / 1 } ,code

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2: is a mnemonic, 3-character code specifying the tape function to be performed.

BSF	Backspace to tape mark.
BSR	Backspace to interrecord gap.
ERG	Erase gap (write blank tape)
FSF	Forward space to tape mark.
FSR	Forward space to interrecord gap.
REW	Rewind tape
RUN	Rewind and unload tape
WTM	Write tape mark

4.1.2.13 Controlling Short Blocks (DMCTL, TRUNC)

The TRUNC option of the DMCTL macro is compatible with the OS/3 TRUNC macro. It is used with blocked output records to write short blocks of data to tape files. Its use is optional with fixed-length blocked records, but required when building variable-length blocked records in the I/O buffer.

Format:

NAME	OPERATION	OPERAND
[label]	DMCTL	{cdibname / (1) / 1} ,TRUNC

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2: identifies the truncation feature.

4.1.2.14 Controlling Skipping to the Next Block (DMCTL, RELSE)

The RELSE option of the DMCTL macro provides the ability to skip over records within a block (on tape) and begin processing the first record of the next block. This feature is compatible with the OS/3 RELSE macro.

FORMAT:

NAME	OPERATION	OPERAND
label	DMCTL	{cdibname / (1) / 1} ,RELSE

Positional Parameter 1: (Same as DMINP macro)

Positional Parameter 2: identifies the block release feature.

4.1.2.15 Controlling Print Forms (DMCTL, SK/SP)

The DMCTL macro offers a forms control option for spacing and skipping of printer forms. Forms motion can occur before, after or both before and after a line is printed.

Format:

NAME	OPERATION	OPERAND
[label]	DMCTL	{cdibname / (1) / 1} , {SK / SP} [,m] [,n]

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2: identifies the print control option. SK indicates form skipping; SP indicates form spacing.

Positional parameter 3: specifies either the number of lines (Ø through 15) for immediate spacing or the channel code (1 through 15) for immediate skipping. The number of lines spaced is determined by the PRAD keyword parameter (the default is an advance of one line).

Positional parameter 4: specifies the number of lines (Ø through 15) for delayed spacing or the channel code (1 through 15) for delayed skipping.

4.1.2.16 Controlling Print Overflow (DMCTL, PRTOV)

The PRTOV option is used to control forms overflow detection and actions. This macro can only be issued if PRINTOV=YES is specified at file OPEN.

FORMAT:

NAME	OPERATION	OPERAND
[label]	DMCTL	{cdibname / (1) / 1} ,PRTOV, [{9 / 12}] ,[{SKIP / REPORT}]

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2: identifies the forms overflow feature

Positional parameter 3:

{9 / 12} specifies the channel which will set the overflow condition (i.e., channel 9 or channel 12)

Positional parameter 4:

SKIP - Upon detecting a forms overflow condition, DM will automatically skip to the home paper position. No indication will be passed back to the user than an overflow condition was encountered. This is the default. (This is identical to not specifying an overflow routine on the PRTOV macro under DTF interfaces.)

REPORT - Upon detecting a forms overflow condition, DM will inform the user by reporting exception status (see the INTERFACE STATUS section of appendix A.1 for additional information.) (This is the substitute for specifying an overflow routine on the PRTOV macro under DTF interfaces.)

4.1.2.17 Controlling Stacker Selection (DMCTL,SS)

The stacker selection option of the DMCTL macro provides the means of controlling card punch output hoppers. If this option is issued, the OPEN RIB keyword parameter CONTROL=YES must have been specified.

The macro will be ignored if issued to a card file that does not support stacker selection.

Format:

NAME	OPERATION	OPERAND
[label]	DMCTL	{cdibname / (1) / 1} ,SS [,{1 / 2}]

Positional parameter 1: (Same as DMINP macro)

Positional parameter 2: identifies the stacker select option.

Positional parameter 3: specifies selection of the output stacker. 1 specifies the normal stacker; 2 is the select (error) stacker. If this parameter is omitted, stacker 2 is the default.

4.1.2.18 Create a Breakpoint to a Spool Output File (DMBRK)

The DMBRK macro creates a breakpoint in a printer or punch spoolfile. The functionality is equivalent to the BRKPT macro (which can only be issued under DTF interfaces).

The macro is ignored if issued in a system that does not have the spooling capability.

The macro should only be issued against a file which is OPEN.

Format:

NAME	OPERATION	OPERAND
[label]	DMBRK	{cdibname / (1) / 1}

Positional parameter 1: (Same as DMINP macro)

4.1.2.19 Tape User Label Processing (DMLAB)

The DMLAB macro is used to process optional user standard or nonstandard tape labels. Control is always returned inline with the appropriate status posted.

FORMAT:

NAME	OPERATION	OPERAND
[label]	DMLAB	{cdibname / (1) / 1}, {workarea / (Ø) / Ø}, {IO / END}

Positional Parameter 1: (same as DMINP)

Positional Parameter 2:

Specifies the workarea (via symbolic address or register notation) that will receive the label (for an input file) or contains the label (for an output file). It must be fullword aligned, and cannot be the same area as the workarea used to process records or any of the file's I/O buffers.

Standard labeled file considerations:

The workarea will contain only the 80 byte label. If block numbering is specified, the workarea must be immediately preceded by a 4 byte fullword aligned field.

Non-standard labeled file considerations:

The format of the workarea will be the conventional variable format, i. e. 4 byte RDW followed by data (i.e. label). The length of the label +4 will be passed in the first 2 bytes of the RDW. The length is user supplied on output and DM supplied on input (and includes the 4 byte RDW). On output, the length of the label cannot exceed the file's buffer (block)size.

If block numbering is specified, the 4 byte RDW will serve as the block number field. On output, upon receiving control back from the DMLAB macro, the record size (in the RDW) will have been overlayed with the block number. Therefore, the record size (in the RDW) must be re-established for each label.

If backwards reading is specified (i.e. READ=BACK RIB specification), the size of the workarea must be at least buffer (block) size +4.

Positional Parameter 3:

IO - specifies that the next label is to be read (for an input file) or written (for an output file).

When control is returned, the status possibilities are:

- Successful - if the label was successfully read or written. CT$LCODE (in the CDIB) will contain the appropriate code (C'O', C'V' or C'F').
- error - if an error occurred.
- exception - if no more labels can be read or written. The device dependent indicator, CT$LEND, will be set to indicate exception status due to a "label processing termination required" condition. The DMLAB (END) macro must then be issued to terminate label processing. (This is the only condition for exception status being reported on a DMLAB(IO) macro.)

END - specifies that label processing is to be terminated (i.e. no more labels read or written) and that the last non-label related function request (i.e. OPEN, CLOSE, DMFEV, DMINP or DMOUT) is to be completed.

When control is returned, the status that is reported is actually the status of the last function request. The appropriate device independent and device dependent information is available. For example, if the last function request was an DMINP call, the status that is posted on the DMLAB(END) macro is the status of the DMINP call. The status could potentially be exception status due to any of the possible exception conditions (i.e. EOF, label processing required, etc.).

When control is returned, the contents of register Ø will have been changed to reflect the value that was in register Ø when the last function request was issued. For example, if the last function request was a DMINP call and workarea processing is being used, when successful status is returned on the DMLAB (END) call, register Ø will be pointing to the record in the original workarea that was presented on the DMINP call.

4.1.3 Device Independence

CDI provides the user with a number of tools to achieve a high degree of device independence. A standardized set of declarative and imperative directives are supplied to describe and control the processing of user data without directly associating the data to a physical device.

4.1.3.1 Describing File Characteristics

The CDI Resource Information Block (RIB) declarative macro allows the user to describe:

1. the attributes of files
2. processing action to be taken under selected conditions.

4.1.3.1.1 Defining File Attributes

The keyword parameters which define file attributes (e.g. RCSZ, RCFM, BFSZ, etc.) can be defined independent of the physical device on which the file will be processed. However, the data management system does not absorb any differences between device capacity and logical records which exceed this capacity. For example, a 200-byte record could be output to tape or disk, but not to a card punch or printer without truncating the data; data management will not, in this case, produce multiple punched cards or print lines. Therefore, the user is responsible for selecting the devices for which the defined file attributes apply.

Certain combinations of logical file attributes and physical devices cause differences in I/O buffer size requirements and will have an adverse effect on device independence capabilities. For example, an 80-byte record size is appropriate for output to a printer or a disk. However, each of these devices requiresa different minimum buffer space: 80 bytes for printer; 512 bytes for disk. If the I/O buffer space is allocated local to the calling program (specification of IOA1 keyword), then the program would be required to define a 512-byte buffer to accomodate both devices; however, Consolidated Data Management will resolve these differences through dynamic buffer allocation.

CDM will identify the device, calculate the required minimum buffer space for the file, and allocate the space to the file for subsequent processing. The calling program presents/receives its data in a workarea, and need not specify a buffer size (BFSZ) or I/O buffer location (IOA1) within the program (See Appendix D.).

4.1.3.1.2 Defining Processing Attributes

Keyword parameters which identify actions to be taken during processing are, in general, device and/or access method dependent. When this category of keyword parameters is specified, the appropriate action will be taken only when the device/access method to which it applies was assigned to the file at file open; otherwise, the parameter will be ignored. This facility allows several device-dependent actions to be associated with a single file definition. Device assignment at execution time provides the selection criteria for each specification.

4.1.3.2 Device Independent File Processing

The CDI imperative macros described in section 4.1.2 can be divided into three categories relative to device independent file processing:

1. those which are common across all devices,
2. those which are not supported by a given access method or device, and
3. those which have no effect on file positioning and do not alter file conditions.

4.1.3.2.1 Device Independent Imperatives

A subset of the CDI imperative macros are truly independent of the access methods or devices supported by the system. These imperatives are limited to the serial processing of data in an input or output mode of operation. For example, an application dedicated to serially reading data would use the

```
DMINP   CDIB-name, workarea
```

form of the imperative macro. No other parameters would be specified (e.g. UNK, KEY, SEQ, RAN, etc.).

A selected set of devices which support input operations (or output operations for DMOUT) and conform to the file attributes defined (see Section 4.1.3.1.1), can be accessed independent of the device assigned through job control at execution-time.

4.1.3.2.2 Invalid Imperative Macros

This category of imperative macro requests a function be performed which is not supported by the access method. Output operation to an input-only device or input operations to an output-only device fall into this category. An invalid imperative macro error condition is reported to the calling program along with the appropriate status indicators.

4.1.3.2.3 Ignored Imperative Macros

Certain imperative functions which are not supported by an access method or device can be ignored and reported as successfully completed functions. These imperatives have no effect on logical file positioning. Macros falling into this category include most of the paper peripheral control functions (skip, space, home, etc.)

Various parameters supported by an imperative macro (e.g. keyed or random specifications on DMINP or DMOUT macros) are also ignored if the access method/device does not support the facility.

INTENTIONALLY

LEFT

BLANK

4.2 Operator Interfaces

There are no operator interfaces associated with the definition of the interface. Operator interfaces that relate specifically to CDI support elements will be defined in their respective documentation.

4.3 Data Bases

The CDIB and RIB structures are the only data bases of the Common Data Interface. Data files and the external processor dependent structures are the data bases of the Data Management processors.

5. ENVIRONMENTAL CHARACTERISTICS

5.1 Hardware Required

The Common Data Interface is intended to be functionally compatible with both SUL and OS/3 hardware. The interface will remain the same while support code, e.g. data management, processors, PIOCS, etc. will be changed to accomodate differences between devices.

5.2 Restrictions

No restrictions are anticipated at this time.

6. AVAILABILITY, RELIABILITY, AND MAINTAINABILITY

This section of the document is not applicable to the definition of the Common Data Interface; no logic is defined, only the interface and its associated macros. Such ARM requirements will be described in other documents that specify the support of the CDI.

7. PERFORMANCE

The interface between the user and the Data Management system is established through the SCALL SVC. The Supervisor SVC decode service is accelerated for all SCALL entries (immediate recognition - no priority scans). Register stack frame manipulation and a shared code linkage complete the path to the Centralized File Manager who, in turn, routes to the appropriate Data Management processor that services the request.

The performance impact on the interface is, therefore, embodied in those services (see the respective documentation for performance considerations).

8. STANDARDS

None.

9. STANDARD DEVIATIONS

None.

10. DOCUMENTATION

The S-2 Product Software Description (PSD), A-41058, was used as a base for the design of the Common Data Interface.

For a study of compatibility issues, the OS/3 Data Management User Guide, UP-8068, and the Multi-Indexed Random Access Method (MIRAM) CPSD, 854, were referenced.

11. SUPPORT

The interface declarative and imperative definitions are provided as type 1 software and will be supported and maintained as other macro library items.

APPENDICES

INTENTIONALLY

LEFT

BLANK

APPENDIX A : COMMON DATA INTERFACE BLOCK (CDIB)

A. 1 CDIB LAYOUT AND FIELD DEFINITIONS

	Ø	1	2	3
Ø	CDIB ID	CDIB LENGTH	FUNCTION CODE	FUNCTION CONTROL 1
4	FILENAME			
8				
12	INTERFACE FLAGS		INTERFACE STATUS	
16	FUNCTION LINK		FUNCTION CONTROL 2	FUNCTION CONTROL 3
2Ø	DEVICE INDEPENDENT			
24	REQUEST DATA & COMPLETION STATUS			
28				
32	DEVICE DEPENDENT			
36	REQUEST DATA & COMPLETION STATUS			
4Ø				

The CDIB DSECT can be generated and used as follows:

```
        VTOC    CDIB=YES
USING       CD$CDIB,1
```

Each of the CDIB fields is described below along with (in parenthesis) the corresponding DSECT name and the number of bytes in the field. (Certain fields can be referenced by more than one name; new names have been provided, in some instances, for clarity purposes. Only the recommended (i.e. new) names are described here. The "old" names are still supported; see the DSECT expansion for all allowable names.)

CDIB ID (CD$ID,1):

contains the CDIB identifier that distinguishes this structure from a DTF. The DSECT name, CD$CDID, is equated to the id value (x'11').

CDIB LENGTH (CD$LGTH, 1):

contains the CDIB length. The DSECT name, L'CD$CDID, is equated to the length value (x'2C').

FUNCTION CODE (CD$FCOD,1):

contains the function code of the function to be performed. It is conditioned by the imperative macros, and is interrogated by the Centralized File Manager and the access methods. DSECT names are provided which are equated to the individual function codes.

FUNCTION CONTROL BYTE 1(CD$FCTL1,1):

contains function control information which further defines the function to be performed. It is conditioned by a few imperative macros which require information (in addition to the function code) to be passed on to the processor.

FILENAME (CD$FNME,8):

contains the logical (LFD) name of the file. It can be up to 8 characters long.

INTERFACE FLAGS (CD$IFLG,2):

contains indicators which the user employs to convey information to data management. The following indicators can be set by the user:

CD$INIT - file initialization. (Honored by open process only.) Tells D.M. to "erase" entire file (i.e. after the open process has completed, there will be no records in the file.) The open process will not turn off this indicator in the CDIB; if the same CDIB is used for future file opens those files will also be erased.

CD$IEXT - file extend (tape files only). (Honored by open process only) Tells D.M. that file is to be extended.

CD$IFCB - FCB-in-core. (Honored by open process only.) Tells D.M. that there is no job control for the file. The open process expects to find the address of the FCB (file control block) in the device independent request data area (CD$FCBAD).

CD$IMSUP -message suppression. (Honored by all imperatives.) Tells D.M. not to display error messages on the console or log.

INTENTIONALLY

LEFT

BLANK

CD$ICMSG - canned message processing. (Honored by DMOUT imperative only.)

If the following 3 conditions exist:

- the CD$ICMSG indicator is set on;
- the imperative is a DMOUT;
- the first character of the user's data (in the user supplied workarea) is a dollar sign

then this special canned message processing is invoked. (If all 3 conditions are not met, no special processing is performed.)

The functionality consists of using a user supplied canned message id to retrieve the corresponding message from the canned message file. Then a new record is "built" (in a D.M. supplied dynamic buffer) using the message as the data. Finally, the DMOUT operation will be performed to the specified file using this new buffer as the workarea.

The following rules apply:

- workarea processing must be employed
- variable length record processing must be employed. The original record which is presented to D.M. must be in variable length format (i.e. 4 byte RDW followed by the user's data). The new record which is built by D.M. will also be in variable length format (64 bytes long).
- the user's data (in the original record) must conform with the buffer format for the supervisor GETMSG macro. (See the Supervisor User Guide UP-8075 for information on the GETMSG macro.) Character insertion is supported.
- The file characteristics must support a 64 byte record.

INTERFACE STATUS (CD$ISTAT,2):

contains indicators which data management employs to convey information to the user. The following indicators can be set by D.M.:

CD$ISUCC - function successful. This indicator should be checked after every OPEN,CLOSE or imperative function to determine whether or not the function was successful.

CD$IOPT - optional file processing active. This indicator will be set on by OPEN if the "resource is unavailable" and the OPTN=YES specification is set in the RIB.

The "resource is unavailable" if:

- o there is no job control for the file, or
- o there is job control, but no pub is available and the user specified OPT on the DVC statement.

This indicator will also be set on for all future imperatives issued against the file. The status that is returned is as follows:

- o If the function is a DMINP, an end-of-file indication will be returned.
- o If the function is anything else but a DMINP, a function successful indication will be returned.

Note: Whenever this indicator (CD$IOPT) is set, there will be no additional information passed back in the device independent or device dependent request data and completion status fields.

CD$IEXC - exception status. This indicator can only be set "on" for unsuccessful imperatives (i.e. it need not be interrogated if the function was successful); its intent is to differentiate between "special conditions" and errors.

Under "normal" (i.e. default) conditions, where special functionality is not requested, the following rules apply:

- for input devices, exception status is only reported when an end-of-file condition is encountered. Therefore, exception status always indicates EOF. (The device independent indicator, CD$EOD (bit 1 of filename C byte 3) is set on to indicate exception status due to an "EOF" condition, but it is not necessary to interrogate it.)
- for output devices, exception status is never reported. Therefore, unsuccessful status always indicates an error condition.

The following "special parameter specifications" (which are not defaults and must be explicitly made) request that other conditions (in addition to EOF) be reported as exception status (see the descriptions of the individual parameters for information on the functionality that they provide):

- PRINTOV=REPORT or YES
- TRUNC=YES
- ULABEL=YES

If one or more of these "special parameter specifications" are made with the result being that exception status can potentially be reported for multiple reasons on input, the indicator CD$EOD must be interrogated to determine EOF.

INTENTIONALLY

LEFT

BLANK

FUNCTION LINK (CD$LINK,2):

contains a relative index to file entry within the Open File Table. It is established at file OPEN and on all future imperatives. It is intended to accelerate the identification of the external control structure.

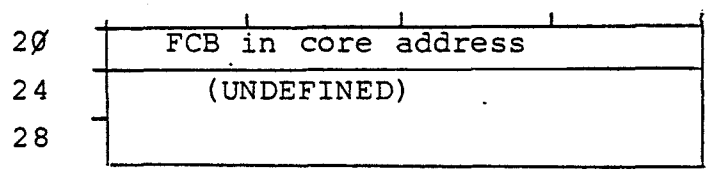
FUNCTION CONTROL BYTES 2 and 3 (CD$FCTL2 and CD$FCTL3, 1 each):

contains additional function control information when more than 1 byte (CD$FCTL1) is required. (See FUNCTION CONTROL BYTE 1)

DEVICE INDEPENDENT REQUEST DATA AND COMPLETION STATUS (CD$REQD and CD$CS,12):

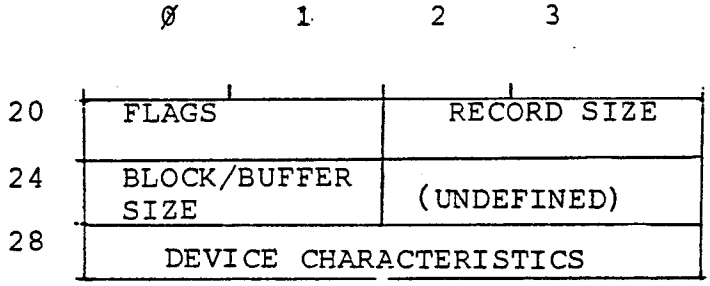
A "bi-directional" field that is used to pass device independent (i.e. common for all access methods) information back and forth between the user and D.M.

(a) REQUEST DATA is the term to describe information passed to DM from the user. Currently the only defined use is at OPEN time where the 4 byte FCB in core address is passed to DM in the first word (CD$FCBAD). (See INTERFACE FLAGS section).



(b) COMPLETION STATUS is the term to describe information passed to the user from D.M. The information will vary depending on whether the function was OPEN or NON-OPEN, and also on whether or not the function was successful.

(1) SUCCESSFUL OPEN IMPERATIVE



o FLAGS (CD$OFLG,2):

CD$EXST - existing file. File was previously created and has been opened for purposes of extending, reading or updating. This indicator will be set on for a tape file if the file type is OUTPUT with extend specified, or INPUT. It will be set on for a disk (MIRAM) file if the file was previously created.

CD$SATFL - SAT type file
CD$FIXBK - fixed length blocked record format
CD$VARBK - variable length " " "
CD$FIXUN - fixed length unblocked " "
CD$VARUN - variable length " " "
CD$UNDEF - undefined record format
CD$READ - reads allowed to device
CD$WRITE - writes allowed to device

(NOTE: For DISK (MIRAM) files the record format is treated as unblocked; so either CD$FIXUN or CD$VARUN will be set.)

- RECORD SIZE (CD$ORCSZ, 2):

  contains the effective record size. This is the size of the work area that would have to be used to hold the record. For variable length records, this reflects the maximum record size (including the 4 byte record header).

- BUFFER/BLOCK SIZE (CD$OBFSZ,2):

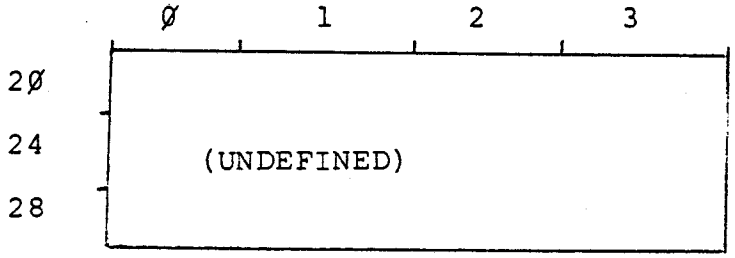
  contains the effective buffer/block size. This is the minimum size of the I/O buffer. It includes any necessary overhead for variable length block and record headers, control bytes and padding (to force an even number of bytes).

- DEVICE CHARACTERISTICS (CD$ODEV,4):

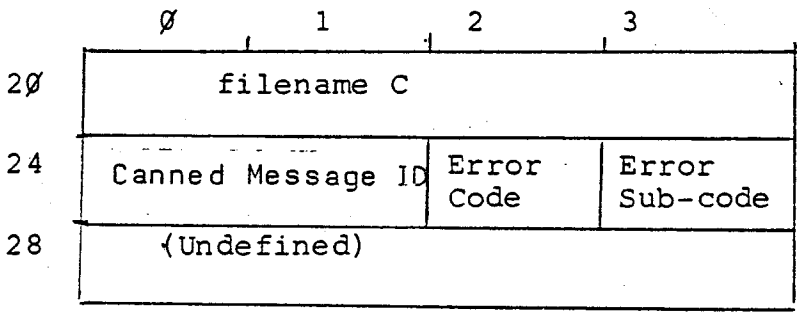
  contains the device type (1 byte), sub-type (1 byte), and feature bits (2 bytes) for the associated PUB. DSECT names are provided for the various device types. A full description of these 4 bytes can be found in the Supervisor/ Job Control working paper, WP378.

(2) SUCCESSFUL NON-OPEN IMPERATIVE

This field is undefined for successful NON-OPEN imperative functions.



(3) UNSUCCESSFUL IMPERATIVE (OPEN or NON-OPEN)



o FILENAME-C (CD$FNMC,4):

contains the error flags that under DTF interfaces were referred to as the filenameC flags. These 4 bytes are detailed at the end of this appendix. The individual bits have the same meaning as they did under DTF interfaces. In order to conform with the filenameC format under DTF interfaces, the meanings of the individual bits were not changed. Therefore, in some instances, a bit will have different meanings dependent on the device; but, in general, the indicators are device independent.

o CANNED MESSAGE ID (CD$CMID,2):

contains the canned message id of the message which corresponds to the error code. The id is placed in this field regardless of whether or not message suppression is invoked.

If there is no message for the particular error code, this field will be zero (i.e. X'ØØØØ').

If the canned message id is to be used to display the message to a particular device, one should be aware that most DM messages employ character insertion (to fill in the variable portions of the message). Those that do all use the same inserts in the following order:

- o 8 character LFD name
- o 3 character device address
- o 2 character error sub code

(See the description of the canned message processing indicator, CD$ICMSG, in the INTERFACE FLAGS section for information on a CDI canned message processing facility.)

Notes:

- o The supervisor GETMSG facility will ignore any specified inserts if the particular message does not support any.
- o The 3 character device address should probably be left as blanks because the appropriate value would be difficult to determine.
- o The 1 byte error sub code that is available at CD$SCOD must be converted into unpacked format (2 bytes).

- ERROR CODE (CD$ECOD,1):

  contains the D.M. error code. The error code is reflected in the corresponding message (if one exists for the particular error code) which is displayed on the console (unless message suppression is invoked). All D.M. messages begin with DMXX where XX is the error code.

  The following error codes do not have corresponding messages:

  X'31' : random record not found (disk files only)

  X'34' : exception condition (i.e. EOF, forms overflow, input record truncation, etc.)

- ERROR SUB CODE (CD$SCOD,1):

  contains the D.M. error sub code which is used to further define the particular error for those error codes which are not specific enough.. The sub code is also reflected in the message. Messages with sub codes end with TYPE=ZZ where ZZ is the error sub code.

DEVICE DEPENDENT REQUEST DATA AND COMPLETION STATUS(CD$DDINF,12):

A "bi-directional" field that is used to pass device dependent information back and forth between the user and DM. (The DSECT names employ different prefixes depending on the particular device.)

(a) REQUEST DATA: (undefined)

(b) COMPLETION STATUS:

(1) PRINTER

- SUCCESSFUL OPEN IMPERATIVE

o FORMS OVERFLOW AND LENGTH (CP$PAGE,2):

Forms overflow line count (in first byte) and total line count (in second byte).

o PUB ADDRESS (CP$PUB,2):

absolute address of PUB

(2) CARD

- SUCCESSFUL OPEN IMPERATIVE

o INPUT BUFFER SIZE (CR$IBFSZ,2):

For combined card files, this field represents the size of the input buffer, and CD$OBFSZ (in the device independent area) would contain the size of the output buffer.

(3) DISK (MIRAM)

- SUCCESSFUL OPEN IMPERATIVE

  o FLAGS (2):

    CM$INDEX - indexed operations are permitted.
    CM$RCB - record control byte is specified.

  o INDEX BUFFER SIZE (CM$INDS,1):

    number of 256-byte sectors in index buffer.

- SUCCESSFUL or UNSUCCESSFUL NON OPEN/CLOSE IMPERATIVE

  o KEY INFORMATION (CM$KEYIN,2):

    BYTE Ø BIT Ø DUPLICATE KEY ERROR
    1 KEY CHANGE ERROR
    2 (undefined)
    3 KEY 5
    4 KEY 4
    5 KEY 3
    6 KEY 2
    7 KEY 1

    BYTE 1 BIT Ø DUPLICATE KEY AHEAD
    1 DUPLICATE KEY CREATED
    2 KEYED RECORD LAST INPUT
    3-7 (undefined)

(4) TAPE

- SUCCESSFUL OPEN IMPERATIVE

  o FLAGS (1)

    CT$BKNO-block numbering specified

- UNSUCCESSFUL IMPERATIVE (ALL)

  o FLAGS (1)

    CT$LABEL - exception status due to a "label processing required"condition.

    CT$LEND - exception status due to a "label processing termination required" condition.

    CT$INPUT - input label processing is required (if indicator is off, it means that output label processing is required). This indicator should only be interrogated if CT$LABEL is on.

- CT$LABEL is on or SUCCESSFUL DMLAB (IO)

  o LABEL CODE (CT$LCODE,1):

    C'O' if file or volume open in progress
    C'V' if volume close in progress
    C'F' if file close in progress

INTENTIONALLY

LEFT

BLANK

A.2 FILENAMEC ERROR FLAGS

filenameC BYTE Ø

BIT \ SC PROCESSOR	MIRAM	TAPE	READER PUNCH	PRINT
Ø	last block on track accessed	reserved	record size invalid	line truncated
1	invalid ID	reserved	reserved	invalid control character
2	invalid control structure	invalid control structure	validity check	character mismatch
3	hardware error	hardware error	hardware error	hardware error
4	error found in OPEN	error found in OPEN	error found in OPEN	error found in OPEN
5	error found in CLOSE	error found in CLOSE	error found in CLOSE	error found in CLOSE
6	invalid macro sequence	invalid macro sequence	invalid macro sequence	invalid macro sequence
7	file lock error	reserved	reserved	record size invalid

NOTE: The term reserved refers to bits that are not supported or illogical for that device/processor; reserved bits will not be set.

filenameC BYTE 1

BIT \ SC PROCESSOR	MIRAM	TAPE	READER PUNCH	PRINT
ø	I/O completed	I/O completed	I/O completed	I/O completed
1	unrecover-able error	unrecover-able error	unrecover-able error	unrecover-able error
2	unique unit error	unique unit error	unique unit error	unique unit error
3	record not found	reserved	reserved	reserved
4	unit exception	unit exception	unit exception	unit exception
5	wrong length found	reserved	reserved	reserved
6	end of track	reserved	reserved	reserved
7	end of cylinder	reserved	reserved	reserved

filenameC BYTE 2

BIT \ SC PROCESSOR	MIRAM	TAPE	READER PUNCH	PRINT
ø	command reject	command reject	command reject	command reject
1	intervention required	intervention required	intervention required	intervention required
2	output parity check	BUS out check	BUS out check	BUS out check
3	equipment check	equipment check	card jam	equipment check
4	data check	data check	data check	data check
5	overrun	overrun	overrun	overrun
6	STOP state	word count zero	STOP state	STOP state
7	device check	data converter check	device check	device check

filenameC BYTE 3

BIT \ SC PROCESSOR	MIRAM	TAPE	READER PUNCH	PRINT
ø	invalid record size	invalid record size	reserved	reserved
1	logical end of file	logical end of file	logical end of file	forms overflow
2	logical end of volume	logical end of volume	reserved	reserved
3	processing inhibited	wrong length error	reserved	reserved
4	invalid index condition	reserved	reserved	reserved
5	input truncation	input truncation	input truncation	input truncation
6	duplicate key	reserved	reserved	reserved
7	reserved	reserved	reserved	reserved

A.3 METHODS FOR USING BIT AND BYTE DSECT EQUATES

All CDIB DSECT equates for bit (i.e. flag) indicators and byte values employ the following format:

```
CD$ISUCC      EQU  *,X'8Ø'
CD$IMSUP      EQU  *,X'1Ø'
CD$OUT        EQU  *,X'2Ø'
```

The following 2 methods can be used to test a bit indicator:

```
(1)  TM   CD$ISUCC,L'CD$ISUCC
     BZ   EXCEPTION ROUTINE

(2)  IF   CD$ISUCC,*OFF
        EXCEPTION ROUTINE
     ENDIF
```

The following 2 methods can be used to set or clear a bit indicator:

```
(1)  OI   CD$IMSUP,L'CD$IMSUP
     NI   CD$IMSUP,X'FF'-L'CD$IMSUP

(2)  SETS#    CD$IMSUP,*ON
     SETS#    CD$IMSUP,*OFF
```

The following 2 methods can be used to test a byte value:

```
(1)  CLI  CD$OUT,L'CD$OUT
     BE   DMOUT PROCESSING ROUTINE

(2)  IF#  CD$OUT,*EQ/LI,L'CD$OUT
          DMOUT PROCESSING ROUTINE
     ENDIF#
```

A.4 CONDITION CODE SETTINGS

Upon receiving control back from any imperative (i.e. OPEN, CLOSE, DMOUT, DMINP, etc.), the PSW condition code will be set, in accordance with the status of the particular operation, as follows:

FUNCTION STATUS	CONDITION CODE	BINARY MASK
SUCCESSFUL	Ø	8
UNSUCCESSFUL/ERROR	1	4
UNSUCCESSFUL/EXCEPTION (i.e. EOF, forms overflow, input truncation, etc.)	2	2

These settings can be used instead of testing the interface status indicators (i.e. CD$ISUCC and CD$IEXC).

The following 2 methods employ the PSW condition code settings:

(1)
```
DMINP  (1),(Ø)
BC      4,ERROR PROCESSING ROUTINE
BC      2,EXCEPTION PROCESSING ROUTINE
SUCCESS PROCESSING ROUTINE
```

(2)
```
DMINP (1), (Ø)
BAL   R14,STATUS TESTING ROUTINE
SUCCESS PROCESSING ROUTINE

STATUS TESTING ROUTINE

BCR   8,R14
BC    2,EXCEPTION PROCESSING ROUTINE
ERROR PROCESSING ROUTINE
```

APPENDIX B: RESOURCE INFORMATION BLOCK (RIB)

B.1 RIB EXAMPLES

EXAMPLE #1

RIB equates:

```
RB$BFSZ     EQU     X'Ø4Ø2',2       BFSZ
RB$IOA1     EQU     X'ØBØ3',3       IOA1
RB$KEY1     EQU     X'14Ø4',4       KEY1 PARAM
RB$KLC      EQU     *,2              -KEY LOC
RB$KLN      EQU     *,1              -KEY LEN
RB$KSP      EQU     *,1              -KEY SPECS
RB$NCND     EQU     X'ØØ'            -NCHG/NDUP
RB$DNC      EQU     X'8Ø'            -DUP/NCHG
RB$CND      EQU     X'4Ø'            -CHG/NDUP
RB$CD       EQU     X'CØ'            -CHG/DUP
RB$MODE     EQU     X'1901',1       MODE
RB$SEQ      EQU     X'08'            -SEQ
RB$RAN      EQU     X'ØØ'            -RAN
RB$RANH     EQU     X'Ø4'            -RANH
RB$WORK     EQU     X'3101',1       WORK
RB$NO       EQU     X'ØØ'            -NO
RB$YES      EQU     X'01'            -YES
```

RIB CALL:

```
RIB BFSZ=256,IOA1=BUFF, WORK=YES
    MODE=RAN, KEY1=(1Ø,26,NDUP,NCHG)
```

RIB STRUCTURE: (generated by RIB macro)

```
DC    AL2(RB$STRT)                  RIB START
DC    AL2(RB$BFSZ)                  BFSZ HIC
DC    AL(L'RB$BFSZ)(256)            BFSZ DATA
DC    AL2(RB$WORK)                  WORK HIC
DC    AL(L'RB$WORK)(RB$YES)         WORK=YES
DC    AL2(RB$IOA1)                  IOA1 HIC
DC    AL(L'RB$IOA1)(BUFF)           IOA1 ADDRESS
DC    AL2(RB$MODE)                  MODE HIC
DC    AL(L'RB$MODE)(RB$RAN)         MODE=RAN
DC    AL2(RB$KEY1)                  KEY1 PARAM
DC    AL(L'RB$KLC)(26)              KEY LOC. SPEC
DC    AL(L'RB$KLN)(10)              KEY LEN SPEC
DC    AL(L'RB$KSP)(RB$NCND)         KEY SPECS
DC    AL2(RB$REND)                  RIB END
```

EXAMPLE #2

RIB CALL:

```
RIB  BFSZ=.TAG1,IOA1=.TAG2,WORK=.TAG3,
     MODE=.TAG4,KEY1=.TAG5
```

RIB STRUCTURE: (generated by RIB macro)

```
DC    AL2(RB$STRT)              RIB START HIC
DC    AL2(RB$BFSZD)             BFSZ HIC(INDIRECT)
DC    AL(L'RB$BFSZD)(TAG1)      ADDR of BFSZ SPEC.
DC    AL2(RB$WORKD)             WORK HIC (INDIRECT)
DC    AL(L'RB$WORKD)(TAG3)      ADDR OF WORK SPEC.
DC    AL2(RB$IOA1D)             IOA1 HIC (INDIRECT)
DC    AL(L'RB$IOA1D)(TAG2)      IOA1 ADDR(INDIRECT)
DC    AL2(RB$MODED)             MODE HIC (INDIRECT)
DC    AL(L'RB$MODED)(TAG4)      ADDR of MODE SPEC.
DC    AL2(RB$KEY1D)             KEY1 HIC(INDIRECT)
DC    AL(L'RB$KEY1D)(TAG5)      ADDR of KEY1 SPECS
DC    AL2(RB$REND)              RIB END HIC
```

INDIRECT DATA: (user generated):

```
TAG1    DC    AL(L'RB$BFSZ)(256)
TAG2    DC    AL(L'RB$IOA1)(BUFF)
TAG3    DC    AL(L'RB$WORK)(RB$YES)
TAG4    DC    AL(L'RB$MODE)(RB$RAN)
TAG5    DC    AL(L'RB$KLC)(26)
        DC    AL(L'RB$KLN)(1Ø)
        DC    AL(L'RB$KSP)(RB$NCND)
```

B.2 RIB BASE/DISPLACEMENT

The Base/Displacement feature is a form of address relocation that only pertains to addresses generated within the RIB structure. Its purpose is to accommodate Base Ø and Key Ø reentrant shared code modules that must handle different specifications for a pre-assembled RIB residing in the shared code module. The Base/Displacement facility is invoked by specifying the following keyword parameter:

BASE=(Rx,TAG)

where: Rx is a register number from Ø to 15
TAG is a symbolic address, usually a DSECT name.

For keywords that specify symbolic address the RIB generator would turn out non-relocatable ADCONs by describing the symbolic addresses as an expression:

AL3 (symbolic address-TAG)

All symbolic addresses become fixed displacements off a symbolic base address (TAG). At execution time Rx is loaded by the data management user and represents the base address. When the Data Management control system encounters an address within the RIB, it adds the contents of Rx to the generated displacement in order to arrive at the true address.

An example of how a RIB macro call would appear in a shared code module is given below:

```
            RIB   BASE=(11,DYNMEM),MOD=RAN,RCB=NO,
                  IOA1=.BUFADDR,BFSZ=.SIZE
            .
            .
            .
DYNMEM      DSECT
BUFF        DS   F
BUFADDR     EQU BUFF+1
SIZE        DS   H
            .
            .
            END
```

All keyword specifications that could change must be specified indirectly as are IOA1 and BFSZ in the example. Indirect specification states that the actual specifications can be found at locations BUFADDR and SIZE. At assembly time BUFADDR and SIZE are non-relocatable displacements of 1 byte and 4 bytes. At execution time the shared code module would acquire dynamic memory and load its address in register 11. Before

presenting the RIB to Data Management, the shared code module must determine the buffer address and size, then store it in dynamic memory. The code would look like this:

```
        USING DYNMEM, 11
        ST R6,BUFF      R6=buffer address
       STH R5,SIZE      R5=buffer size
```

Aligning BUFF on a word boundary is an implementation consideration for the user. Data management assumes all addresses are presented in 3 byte fields.

B.3 Dynamic RIB generation

Dynamic RIB generation is the concept of building a RIB structure without calling the RIB macro. The intent here is not to provide a step by step procedure, but identify tools/aids for such an application.

Compilers, for example, must include RIBs in the generated object code to satisfy their users file processing requirements. One method would be to pre-assemble a RIB with all the keyword options supported by the language for that device. These keywords would all be indirect specifications. At compilation time, the compiler would scan tables representing his user requests and translate them into the appropriate data management specifications.

Compilers that are extremely sensitive to object code size may not view the pre-assembled RIB as a very efficient method for two reasons:

(1) indirect specifications require three more bytes than do direct specifications.

(2) a default specification requires an entry in the pre-assemble RIB structure.

The alternative for compilers sensitive to object code size is to build the RIB structure themselves. The RIB equates in the RIBEQU proc is an extremely useful tool for this very special application.

B.4 RIB PARAMETERS FOR DMAPY/DMQRY

ALLOWABLE DMAPY/ DMQRY RIB PARAMETERS

DISK (MIRAM)		TAPE		CARD(READER/PUNCH)		PRINTER	
DMAPY	DMQRY	DMAPY	DMQRY	DMAPY	DMQRY	DMAPY	DMQRY
WORK	NUMREC	WORK	(NONE)	WORK	(NONE)	WORK	(NONE)
IORG	KEY 1	IORG		IORG		IORG	
IOA1 (1)	KEY 2	IOA1 (1)				IOA1 (2)	
SKAD	KEY 3	CLRW					
MODE	KEY 4	READ					
RETR	KEY 5	TYPEFLE(3)					
PROC							
INDA (1)							
KARG							
BFSZ (1)							

NOTES:

(1) The RIB specification AUTOIO=YES must be specified at file OPEN in order to change these parameter specifications.

(2) In order to change this address, neither work area processing nor double buffering can be specified.

(3) One would change the file type from INPUT to OUTPUT to truncate the file where the tape is positioned and then extend it. One would change the file type from OUTPUT to INPUT to read part or all of the file.

APPENDIX C: "LABEL" PARAMETERS

C.1 Disk (MIRAM) "LABEL" Parameters

When a disk (MIRAM) file is created, certain keyword specifications are "saved" in the format label. On all future OPENs of the file, the user specified value is compared against the value in the label. If the values do not match, the file is not opened and an appropriate error condition is returned. (Note: The exception to this rule is that the values in the label for the RCB, INDOUT and SCSZ parameters are always used regardless of the user specification.)

The "LABEL" parameters are:

** BFSZ	**KEYn	**RCSZ
** INDOUT	**RCB	**SCSZ
* INDS	**RCFM	*VMNT

* If these keywords are defaulted (i.e. not specified), the following values are always used:

INDS=Ø

VMNT=NO

**If these keywords are defaulted, the defaults are determined as follows:

- for a "new" file (i.e. newly allocated or INITed), the following values are used:

  BFSZ= "minimum allowable value" (See BFSZ parameter for the algorithm used to determine the minimum value)
  KEYn= (Ø,Ø)
  RCB= NO
  RCFM= FIX
  RCSZ= 255; if RCFM=FIX and RCB=YES
  256; otherwise
  (The purpose of the default RCSZ being 255 or 256 is to insure that the slot size is 256.)
  SCSZ= 256

- for an "existing" file (i.e. to be extended, read, or updated), the values that were saved in the label will be used.

(NOTE: The buffer size that is used at file creation is not saved in the label because this size can vary as long as it is at least as large as the minimum allowable value. If defaulted, this minimum value will be used. See BFSZ parameter for the algorithm used to determine the minimum value.)

C.2 TAPE "LABEL" PARAMETERS

When a tape file is created with standard labels, certain keyword specifications are "saved" in the tape labels. When the file is opened for extension, the user specified value is compared against the value in the label. If the values do not match, the file is not opened and an appropriate error condition is returned.

The "LABEL" parameters are:

BFSZ
RCFM
RCSZ

If these keywords are defaulted (i.e. not specified), the defaults are determined as follows:

- for a "new" file (i.e. file type is OUTPUT and extend is not specified), the following values are used:

  BFSZ=256
  RCFM=FIXUNB
  RCSZ=256

- for an "existing" file (i.e. file type is INPUT, or OUTPUT with extend specified) the values that were saved in the label will be used.

NOTE: If the file does not have standards labels, the "new" file defaults will always be used(if the keyword is defaulted) regardless or whether it is a "new" or "existing" file.

APPENDIX D: DYNAMIC BUFFERING

Currently there is no designed scheme for providing dynamic buffering; users must allocate their own buffers. Therefore, a RIB must always be used at file OPEN and the IOA1 address must be provided.

INTENTIONALLY

LEFT

BLANK

APPENDIX E: DISK FILE SHARE

E.1 INTRODUCTION

Disk file shareability under CDI ranges from block level shareability (the smallest granularity) to exclusive usage (no shareability) with several intermediate levels.

The file share description is organized in two parts. The first part (Section E.2) is optional reading that provides background information. It identifies terms such as logical access path, logical file, physical file, etc. that are used later. The second part (Section E.3) explains file share in terms of the four environments supported under CDI. Each environment describes the file share functionality it provides the end user.

E.2 BACKGROUND

A disk file is said to be shared when information on the physical storage medium is concurrently accessed by several users. This information could be as small as a physical block that may contain one or more user data records. Or, it could be a structure as large as an index that may contain tracks/cylinders full of interrelated data pointers used by data management's search-by-key mechanism in locating a requested user data record. Information on the physical storage medium may be accessed for modification (WRITE capability) as well as for informational (READ capability) purposes. This collection of related/organized information on the disk storage medium is called a physical file.

The DVC-LFD device assignment set makes the connection between physical file name (LBL name) and logical filename (LFD name). All user references to the file are made by logical filename. The data management user requests the use of the file by OPENing a logical filename. The data management user accesses the file by issuing the D.M. imperatives (DMINP, DMOUT, etc.), in conjunction with logical arguments such as relative record number and key values, against a logical filename. The user request for data is in logical terms. Data Management performs a logical-physical transformation in order to deliver the desired data from the disk storage medium. The data management user terminates use of the file, through the logical filename, by CLOSE ing a logical filename.

The OPEN imperative, in addition to requesting use of a physical file, develops the necessary linkages and control structures to perform the logical/physical transformation mentioned above. This collection of related mechanisms is called the logical Access Path (LAP). A LAP is assigned one of the five ACCESS specifications - EXC, SRDO, EXCR, SRD and SADD. If the ACCESS parameter is not specified in the Resource Information Block (RIB) or the // DD job control statement, the default (EXC) is assumed. The ACCESS specification conveys two pieces of information (1) the READ/WRITE capability of the LAP, (2) the level of shareability the LAP can tolerate. This information is used by OPEN to create environments and determine who can participate in those environments.

E.3 ENVIRONMENTS

An environment is established and maintained for physical files. A request to use a physical file (i.e., OPEN imperative) encounters one of two conditions. Either the file is not in use in which case no LAPs exist for it. Or the file currently is in use and one or more LAPs already exist for it. For the not-in-use case, the ACCESS specification of the requesting LAP establishes the environment. Otherwise, the ACCESS specification is used to determine whether the requesting LAP can participate in the environment already established. A requesting participant whose ACCESS specifications is incompatible with an established environment is either waited or returned an error. (depending on the NWAIT RIB parameter specification).

A user terminating the use of a physical file through a particular LAP (i.e., CLOSE) causes the environment to be re-evaluated, enabling previously waited requestors (OPENs) to either establish new environments or participate in a newly-created one. Environments are dynamic and evaluated on every OPEN and CLOSE imperative.

A disk file can be processed in one of four environments outlined below. Each environment is described by listing the following: (1) the creating and participating ACCESS specifications (2) the advantages of processing in the environment, (3) user discipline required by the environment when applicable.

- EXCLUSIVE environment

  - o This environment is created by a LAP with an ACCESS specification of EXC. There are no participating ACCESS specifications.

  - o The LAP has exclusive use of the file for the OPEN to CLOSE duration. No shareability exists in this environment. Only one LAP can exist at any point in time and it has READ/WRITE capability to the file.

- READ ONLY environment

  - o This environment is created by a LAP with an ACCESS specification of SRDO. Participating LAPs must have the SRDO specification.

  - o This environment enables a number of users (LAPs) to share the file for read only purposes. Writing to the file in this environment is prohibited.

- SINGLE WRITER environment.

  o This environment is created by a LAP with an ACCESS specification of EXCR or SRD. If created by the EXCR, any LAP with an SRD specification can participate. When created by SRD, all other LAPs assigned the SRD specification and one LAP with the EXCR specification can participate.

  o This environment restricts the read/write capability to a single user (LAP with the EXCR specification) while other users (LAPs with SRDs) concurrently access the file for read only purposes.

  o Records added to the file by the writer are immediately available to the readers sharing the file.

  o AUTOIO=YES is automatically set by D.M. to insure data integrity. (See the AUTOIO RIB parameter for additional information and considerations.)

- MULTIPLE WRITER environment

  o This environment is created by a LAP with an ACCESS specification of SADD. All participating LAPs must have the SADD specification.

  o Each LAP in this environment has the read/write capability and can update, add, and read records from the file.

  o In this environment a DMINP with intent to update or a DMOUT imperative locks a set of physical blocks to a job, making them unavailable to other jobs. The number of blocks locked is dependent upon the user's buffer size. The blocks locked by the DMINP with intent to update are held until released by the follow-up DMUPD or DMDEL imperatives. This prevents two jobs from concurrently modifying the same physical blocks, thus avoiding 'lost updates.' The block lock generated by the DMOUT imperative is temporary and held only for the duration of the output function. This prevents two jobs from concurrently overwriting the same physical blocks, thus avoiding 'lost records.'

- A task requesting to lock blocks held by another job is waited until those blocks become available. But locked blocks are always made available to jobs that request them for informational purposes.

- Only one set of blocks can be locked to a job at any one time. An attempt to accumulate block locks causes the automatic release of any block locks previously held by the job. An attempt to modify released blocks is reported as an error.

- Block locks are released by:

  - the DMUPD, DMDEL imperative. Releases block locks generated by the DMINP with intent to update.

  - the completion of the DMOUT.

  - dead lock avoidance mechanism. Occurs when job attempts to accumulate block locks.

  - CLOSE imperative that terminates the use of the file through a LAP that acquired block locks currently held by the job.

- Block locks are <u>not</u> released by:

  - the DMINP for informational purposes.

  - any positioning imperatives such as the DMSEL.

  - any imperatives, including DMOUT and DMINP with intent to update, issued against other LAPs with an ACCESS specification other than SADD. An example of this is when a job is accessing two physical files A and B. A thru a LAP with a SADD specification and B thru a LAP with an EXCR specification. Neither a DMINP with intent to update nor a DMOUT imperative issued against the LAP with the EXCR specification would release any block locks held by the job.

- Records added to the file are immediately available to other users of the file.

- AUTOIO=YES is automatically set by D.M. to insure data integrity. (See the AUTOIO RIB parameter for additional information and considerations.)