To: Distribution

From: James A. Bush

Date: September 24, 1982

Subject: The mtape_ I/O module: A new user interface for tapes

Send comments on this MTB by one of the following means:

By Multics mail on System M to:
Bush.Multics

By Multics mail on MIT-Multics to:
JABush.Multics

To the forum meeting on System M at:
>udd>m>jab>forums>Multics_tape_I/O (tape)

By U. S. mail to:
James A. Bush
Honeywell Information Systems Inc.
Multics Development Center
Cambridge Information Systems Laboratory
Mail Station MA22
575 Technology Square
Cambridge, Massachusetts 02139

Multics Project Internal working Documentation. Not to be reproduced or distributed outside of the Multics Project.

INTRODUCTION

As detailed in MTB 575, there are many problems associated with tape I/O on Multics. Not the least of these problems is the current user interface which, with the many individual I/O modules involved, presents an arbitrary and inconsistent interface to the user.

Another serious problem with our current tape software is performance or lack thereof. Although some interim performance improvements have been made to the tape software for the MR10.0 release, any significant gain in performance will require that the tape_ioi_ intermediate tape I/O module (Documented in MTB 383) finally be implemented. The implementation of tape_ioi_ is being completed as I write this document and will be released in the MR10.1 general release. However, in order to take advantage of the performance enhancements offered by tape_ioi_, the current tape I/O modules would have to be retrofitted to interface to tape_ioi_ instead of the tdcm_ intermediate I/O module. Because of the user interface problems mentioned above, it has been decided not to proceed with this retrofit. (The exception to this will be the retrofit of the tape_mult_ I/O module for the MR10.1 release, which will give system tape I/O a much needed boost in performance, plus provide a testbed and exposure vehicle for tape_ioi_.)

A new iox_ compatible tape I/O module is being designed to address the problems mentioned above, plus the other problems detailed in MTB 575. This new I/O module, called mtape_ (for Multics tape), will initially complement and eventually replace the existing tape_ansi_, tape_ibm_, tape_nstd_ and tape_mult_ tape modules, plus provide a first time support for GCOS, CP5/6, and user defined tape formats as well.

OVERVIEW

The mtape_ I/O module offers a significant departure from the current group of tape I/O modules. Features of mtape_ include:

- o One tape module for all tape formats.

  Unlike the current tape I/O modules, mtape_ will allow processing of several different types of formated tapes including: ANSI, IBM, Multics, GCOS, CP5/6, UNLABELED, and RAW formats. For tape input, information returned from RCP after a successful tape mount is used to determine the appropriate tape format. For tape output, a simple attach description control arg ("-label") is used to designate the desired tape format. Format specific processing requirements are accomplished by externally callable subroutines known as "Per-Format modules".

- Enhanced performance and better error recovery.

  The mtape_ I/O module will use the tape_ioi_ intermediate module for performing physical tape I/O and will enjoy the performance enhancement resulting from all tape I/O being initiated asynchronously (i.e. multi-buffers being written or read with one I/O). The tape_ioi_ intermediate module will also perform all error recovery, taking advantage of hardware error recovery features available within our tape subsystems. This will ensure that all error recovery will be accomplished in a consistent manner for all tape formats.

- Logical separation of volume and file functions.

  Tape volumes and volume sets will be attached and detached, where as tape files will be opened and closed. This is made possible by extensions to the iox_ I/O system (see below) which allows an "open description" to be passed in the open call for defining file specific parameters, instead of passing those parameters in the attach description. This will obsolete the "retention" misfeature perpetrated by the tape_ansi_ and tape_ibm_ tape I/O modules.

- A system of sensible defaults.

  To make the user interface simpler, all values required in an attach or open description will have a reasonable default value assigned. All default values are stored in the users default value segment ([user name].value in the users home directory) and are created at first reference by mtape_. To meet special needs, a user may change these default values, by using the value_set command.

- A method to enable users to define their own tape formats.

  Although mtape_ provides support for all "popular" labeled tape types, there may be circumstances where a user or a site may need to read in tapes created by other vendors, the formats of which do not conform to any of the popular labeled tape types. Since format specific processing is contained in externally callable "per-format" modules which are found using the standard search path mechanism, a user may write his own per-format module and change his search paths appropriately to find it. This tailored per-format module may be either a substitution of one or more of the standard per-format modules, or by using the "-label" attach description argument, a unique named per-format module may also be specified.

For a detailed discussion of the mtape_ I/O module, see the MPM style mtape_ documentation at the end of this document.

RCP Extensions

In order to fulfill the requirements of mtape_, RCP must be extended to return more information to the caller, after a successful tape mount. This information must include:

o Density the tape is recorded at.
o A numerical value of the label type that RCP identified.
o ASCII representation of the volume name, as recorded in the volume label record.
o An indication of whether Operator authentication was required.
o And if so, what was the authentication code used by the Operator.

These changes will be designed in such a way as to not be incompatible with the existing RCP interface. This will be accomplished by using a different version number in the structure returned by RCP, (defined by the include file rcp_tape_info.incl.pl1).

In addition, RCP must be extended to implement a tape unit exchange protocol. This protocol would allow automatic selection of a different tape unit, if after mounting the requested tape volume, it is determined that the recording density is not within the range of density capabilities of the current tape unit. This protocol should also include an externally callable subroutine interface, so that mtape_ could use it as a last ditch error recovery method. (e.g. If an unrecoverable error exists during tape input, mtape_ would remember the current physical position, and demount the the current tape volume. A different tape unit could be requested and after a successful mount, mtape_ could position to the end of the last good block and attempt to re-read the block in error.)

EXTENSIONS TO IOX_

In order to support mtape_, three (3) new entries must be added to the iox_ I/O system. These new entries are:

o iox_$open_file
o iox_$close_file
o iox_$detach

All of these new entries will accept as one of their arguments, a "description" which will contain a string of arguments to be parsed by the I/O module.

In addition, features must be added to iox_$attach_(ptr name) to recognize the following keywords in an attach description:

attach:
open:
close:
detach:

The character strings that follow each of these keywords will be extracted and saved by iox_ (in an allocated area, with a pointer to this area being initialized in the iocb area) and will be passed to the appropriate new entry (i.e. iox_$open_file, iox_$close_file and iox_$detach) as descriptions, when the corresponding old entry (i.e. iox_$open, iox_$close, and iox_$detach_iocb) is called. These changes are needed to support language I/O in mtape_, without changing language I/O.

For details of the new iox_ entries and the changes to the attach entry points, see the MPM style iox_ documentation below.

Name: iox_

Entry: iox_$attach_name

This entry point is the same as the iox_$attach_ptr entry point except that the I/O switch is designated by name and a pointer to its control block is returned. The control block is created if it does not already exist.

Usage

```
     declare iox_$attach_name entry (char(*), ptr, char(*), ptr,
          fixed bin(35));

     call iox_$attach_name (switch_name, iocb_ptr, atd, ref_ptr,
          code);
```

where:

switch_name
is the name of the I/O switch. (Input)

iocb_ptr
points to the switch's control block. (Output)

atd
is the attach description. (Input)

ref_ptr
is a pointer to the referencing procedure, used by the search rules to find an I/O module. (Input)

code
is an I/O system status code. (Output)

Entry: iox_$attach_ptr

This entry point attaches an I/O switch in accordance with a specified attach description. The form of an attach description is given in "Multics Input/Output System" in Section V of the MPM Reference Guide. If the switch is not in the detached state, its state is not changed, and the code error_table_$not_detached is returned.

The I/O module is located using the current search rules.

Usage

```
     declare iox_$attach_ptr entry (ptr, char(*), ptr, fixed
          bin(35));

     call iox_$attach_ptr (iocb_ptr, atd, ref_ptr, code);
```

where:

iocb_ptr
    points to the switch's control block. (Input)

atd
    is the attach description. (Input)

ref_ptr
    is a pointer to the referencing procedure, used by the search rules to find an I/O module. (Input) (See hcs_$make_ptr for more information about ref_ptr.)

code
    is an I/O system status code. (Output)

Notes

The ref_ptr argument can be used to specify a particular I/O module if one by that name is not already initiated, for example:

```
     call iox_$attach_ptr (iocb_ptr, "discard_",
          addr (my_discard_$my_discard_attach), code);
```

In addition to searching the attach description for the I/O module name to attach, there are four keywords which are searched for and have special meaning to the attach entry points. If anyone or all of the keywords, "attach:", "open:", "close:", or "detach:" are found in the attach description, they are interpreted by the iox_$attach entry points as follows:

The character string that follows each keyword, up to the next keyword or the end of the attach description, is stripped from the given attach description and saved in an allocated area with a pointer to this area being initialized in the iocb area. In the case of the "attach:" keyword, this saved character string becomes the new attach description which will be passed on to the specified I/O module. For the "open:", "close:", and "detach:" keywords, the saved character strings become "descriptions" for the iox_$open_file, iox_$close_file and iox_$detach entry points respectively. If after attaching an I/O switch, a user calls the iox_$open entry point and iox_ has previously saved an "open description" in the manner just described, then the open_file entry of the attached I/O module will be called instead of the open entry, with the saved open description being supplied by iox_ as the required open description. The saved close and detach "descriptions" are handled in a like fashion, by having iox_ intercept the call to the close or detach_iocb entries and forwarding the calls to the close_file and detach entries instead, after supplying the necessary "descriptions" from the saved copies.

Entry: iox_$close_file

This entry point closes an I/O switch. If the switch is not open, its state is not changed, and the code error_table_$not_open is returned.

This entry point differs from the iox_$close entry point due to the addition of the close description argument. For those I/O modules that support the close_file entry, the close description offers a means of providing file closing parameters such as a closing comment, where to position to upon closing etc.

Usage

```
     declare iox_$close_file entry (ptr, char (*), fixed
          bin(35));

     call iox_$close_file (iocb_ptr, cld, code);
```

where:

iocb_ptr
points to the switch's control block. (Input)

cld
is the close description. (Input)

code
is an I/O system status code. (Output)

Entry: iox_$detach

This entry point detaches an I/O switch. If the switch is already detached, its state is not changed, and the code error_table_$not_attached is returned. If the switch is open, its state is not changed, and the code error_table_$not_closed is returned.

This entry point differs from the iox_$detach_iocb entry point due to the addition of the detach description argument. For those I/O modules that support the detach entry, the detach description offers a means of providing detach time parameters such as a resource disposition comment to be sent to the system operator.

Usage

```
     declare iox_$detach entry (ptr, char (*), fixed bin (35));

     call iox_$detach (iocb_ptr, dtd, code);
```

where:

iocb_ptr
    points to the switch's control block. (Input)

dtd
    is the detach description. (Input)

code
    is an I/O system status code. (Output)

Entry: iox_$open_file

This entry point opens an I/O switch. The switch must be attached via an I/O module that supports the specified opening mode, and it must be in the closed state. If the switch is not attached, its state is not changed, and the code error_table_$not_attached is returned. If the switch is already open, the code error_table_$not_closed is returned.

This entry point differs from the iox_$open entry point due to the addition of the open description argument. For those I/O modules that support the open_file entry, the open description offers a means of providing file opening parameters such as file format, block size, record size, etc. The open description also allows the logical separation of attachment of resources, such as tape volumes, with the iox_$attach_name and iox_$attach_ptr entry points, and file specific operations for those I/O modules that deal with multi-file resources.

Usage

```
     declare iox_$open_file (ptr, fixed bin, char (*), bit (1)
          aligned, fixed bin(35));

     call iox_$open_file (iocb_ptr, mode, opd, unused, code);
```

where:

iocb_ptr
is a pointer to the control block. (Input)

mode
is the number assigned to the mode as shown in Table 5-1 in Section V of the MPM Reference Guide, e.g., 1 for stream_input, 2 for stream_output. (Input) Named constant values for these modes are defined in iox_modes.incl.pl1.

opd
is the open description. (Input)

unused
must be "0"b. (Input)

code
is an I/O system status code. (Output)

Name: mtape_

The mtape_ I/O module supports physical and logical I/O to or from magnetic tape volume(s), in any one of several formats, including:

ANSI standard format
IBM standard format
IBM Disk Operating System (DOS) format
Multics standard format
GCOS File and Record Control (FRC) format
GCOS Unified File Access System (UFAS) format
CP5/CP6 standard format
Unlabeled format
Raw format

In addition, facilities exist within mtape_ which will permit a user to define his/her own magnetic tape format.

Entries in this module are not called directly by users; rather, the module is accessed through the I/O system. See the MPM Reference Guide for a general description of the I/O system.

Definition of Terms

For the purpose of this document, the following terms have the meanings indicated.

block
: a collection of characters written to or read from a tape volume as a unit. A block may contain one or more complete records, or it may contain parts of one or more records. A part of a record is a record segment. A block does not contain multiple segments of the same record.

file
: a collection of information consisting of blocks pertaining to a single subject. A file may be recorded on all or part of a volume, or on more than one volume.

file set
: a collection of one or more related files, recorded consecutively on a volume set.

per-format module
: an externally callable subroutine with several

standard entry points. The naming convention for per-format modules is in the form of <volume_type>_tape_io_ where <volume_type> is the character string description of the volume label type as returned by RCP on tape input or requested by the user by the use of the "-label" attach description argument or the default label type on tape output. For a discussion of the definition and use of per-format modules, see "Per-format Module Description" below.

record
related information treated as a unit of information.

volume
a reel of magnetic tape. A volume may contain one or more complete files, or it may contain sections of one or more files. A volume does not contain multiple sections of the same file.

volume set
a collection of one or more volumes on which one and only one file set is recorded.

Attach Description

In addition to the I/O module name, only information relevant to the volume or volume set is supplied in the attach description. For the specification of information pertaining to files and file sets, refer to the section titled "Open Description" below. The attach description is a contiguous character string and has the following form:

```
mtape_ vn1 {-comment vn1_str} vn2 {-comment vn2_str} .......
       vnN {-comment vnN_str} {-control_args}
```

where:

1. vni
is a volume specification. In the simplest (and typical) case, a volume specification is a volume name. Occasionally, keywords must be used with the volume name. For a discussion of volume names and keywords see "Volume Specification" below.

-comment vni_str, -com vni_str
allows the optional specification of a message to be

displayed on the operators console at the time volume vni is to be mounted. The comment text, vni_str, may be from 1 to 64 characters in length and must be quoted if it contains embedded white space.

vn1 vn2 ... vnN
: comprise the volume sequence list. The volume sequence list may be divided into two parts. The first part, vn1 ... vni, consists of those volumes that are actually members of the volume set, listed in the order that they became members. The entire volume set membership need not be specified in the attach description; however, the first (or only) volume set member *must* be specified, because its volume name is used to identify the file set. If the entire membership is specified, the sequence list may contain a second part, vni+1 ... vnN, consisting of potential members of the volume set, listed in the order that they may become members. These volumes are known as volume set candidates. (See "Volume Switching" below.)

2. control_args
   is a sequence of one or more attach control arguments. A control argument may appear only once.

-density N, -den N
: on output, specifies the density at which the volume-set is to be recorded, where N can be 200, 556, 800, 1600, or 6250 bits per inch. If this control argument is not specified on output, then the current default density value will be used (See "Default values" below.). On input, this control argument (or in the absence of the -density control arg, the current default density value) will be used as a "first guess" and will be passed to RCP to aid in determining the density of the tape volume at mount time. However, determination of the correct density setting of a tape volume, is the purview of RCP and a user need not concern himself with it.

-device N, -dv N
: specifies the maximum number of tape drives that can be used during an attachment, where N is an integer in the range $1 \le N \le 63$. (See "Multiple Devices" below.)

-display, -ds
: specifies that the entire attach description, after

it has been parsed and any necessary defaults added, will be displayed on the user_output I/O switch.

-label vol_type, -lbl vol_type
specifies that the volume set to be mounted have volume labels of type vol_type, where vol_type can be one of the following valid supported tape formats:

ANSI, ansi	
IBM, ibm	
Multics, multics	
GCOS, gcos	(for GCOS FRC formated tapes)
UFAS, ufas	(for GCOS UFAS formated tapes)
CP6, cp6	(for CP5 or CP6 standard formated tapes)
unlabeled, ulbl	(for unlabeled tapes)
RAW, raw	(for processing any and all tape formats in a raw, user controlled environment)
<STR>	(for user defined formated tapes)

The vol_type value is used in the Per-Format module selection process. The mtape_ I/O module appends the string "_tape_io_" to the vol_type value in order to form the full name of the Per-Format module to searched for (e.g. if the user specified "-label gcos" in the attach description, then mtape_ would form the full name of "gcos_tape_io_" as the Per-Format module to search for). For user defined formatted tapes, the value of "<STR>" may be representative of the actual format for which a private Per-Format module has been written. (e.g. If a user has written a private Per-Format module for say tapes generated on a UNIVAC computer system, this Per-Format module could be named "univac_tape_io_", and this private Per-Format module could be called into execution by simply specifying a "-label univac" argument in the attach description.) For more details on the Per-Format module selection process, refer to the section titled "Per-Format Module Selection" later in this document.

-no_labels, -no_label, -nlbl
specifies that the user wishes to override or further define the "-label" argument specification. For tape input, if the user specified "-label ibm" but did not have a "-no_labels" specification in the attach

description, then for an unlabeled tape volume, RCP would indicate that the tape volume would indeed be unlabeled and mtape_ would return an error indicating that the tape volume was not of the requested type. By using the "-no_labels" specification, this indicates the tape is unlabeled, but is an IBM unlabeled tape and the ibm_tape_io_ per-format module is called to process the unlabeled tape. For tape output, this indicates to the per-format module specified in the "-label" specification (or the current default), that an unlabeled tape volume is to be processed. For those per-format modules that do not process unlabeled tapes, an error will be returned by the attach call. For more detail on the relationship between the "-label" and "-no_labels" attach description arguments, see the section titled "Per-Format Module Selection" later in this document.

-ring, -rg
specifies that the volume set be mounted with write rings. (See "Write Rings and Write Protection" below.)

-speed N1{,N2,...,Nn}, -ips N1{,N2,...,Nn}
specifies desired tape drive speeds in inches per second, where Ni can be 75, 125, or 200 inches per second. (See "Device Speed Specification" below.)

-track N, -tk N
specifies the track type of the tape drive that is to be attached, where N may be either 9 or 7.

Volume Specification

The volume name (also called the slot identifier) is an identifier physically written on, or affixed to, the volume's reel or container.

If a volume name begins with a hyphen (-), the -volume keyword must precede the volume name. Even if the volume name does not begin with a hyphen, it may still be preceded by the keyword. The volume specification has the following form:

```
-volume vni
```

If the user attempts to specify a volume name beginning with a hyphen without specifying the -volume keyword, an error is indicated or the volume name may be interpreted as a control argument.

Volume Switching

There are four types of file set configurations defined:

single-volume file
a single file residing on a single volume

multivolume file
a single file residing on multiple volumes

multifile volume
multiple files residing on a single volume

multifile multivolume
multiple files residing on multiple volumes

The mtape_ I/O module maintains a linked list of volume set members and potential members or candidates, throughout the time the I/O switch is attached. This linked list of volume set members and candidates is called a volume sequence list and is initially generated from the volume specification(s) within the attach description. A minimal volume sequence list contains only one volume, the first (or only) volume set member. For multi-volume operations, additional volume set members or candidates may be specified and included in the volume sequence list, following the mandatory first volume.

If in the course of an output operation physical end of tape is detected, the I/O module prepares to switch to the next volume in the volume set. An attempt is made to obtain the volume name of the next volume in the volume set from the next entry in the volume sequence list. If the volume sequence list is exhausted, then the user is queried for the next volume name to be mounted. This new volume is then added to the volume sequence list. In either case, volume switching occurs, and processing of the file continues.

If in the course of an input operation, an end of file mark is detected followed by what is identified by the per-format module in control as the end of volume trailer sequence, but is not an indication of the end of the current file, then volume switching is initiated as above. The exception to this is when the multics_tape_io_ per-format module is in control and the end of reel record is identified, if the volume sequence list is exhausted, then an error code of error_table_$end_of_file is returned to the user instead of querying him for the next volume name.

In a like fashion to the linked list of volume set members, the mtape_ I/O module builds and maintains a linked list of file attribute structures as each file is processed or recognized in the course of searching for other files. Among other things, the file attribute structure contains information as to the file identifier, file sequence number and indices to the starting and ending volume set member which contain this file. In the course of opening a file, a search of this linked list of file attribute structures is made to determine if the requested file has already been processed or otherwise recognized during this attachment. If an entry for the requested file is found, then the volume set member on which the file resides is compared to the volume currently mounted. If this match is made then the physical file position on the volume is determined (from information contained in the file attribute structure) and the current volume is positioned to the beginning of the requested file. If an entry for the requested file is found in the linked list of file attribute structures, but the starting volume set member that contains this file is different from the current volume, then volume switching is initiated as described above. If no entry for the requested file is found in the linked list of file attribute structures, then a physical search for the requested file is initiated, starting from the current position of the current volume forward through each file position performing volume switching as above when necessary. As each file is identified, even though it is not the requested file, a file attribute structure is built for it and linked into the chain of other file attribute structures.

Multiple Devices

If a volume set consists of more than one volume, the -device N control argument can be used to control device assignment, where N specifies the maximum number of tape drives that can be used during this attachment. N is an integer in the range $1 \leq N \leq 63$. Drives are assigned only on a demand basis, and in no case does the number actually assigned exceed the device limit of the process. The default for an initial attachment to a file in a file set is N equals 1; the default for a subsequent attachment to that (or any other) file in the file set is N equals the previous value of N.

Device Speed Specification

The -speed control argument is used to specify acceptable tape device speeds in inches per second. The module only attaches a device that matches a speed specified by this control argument. If more than one speed is specified, the module attaches a device that matches one of the speeds. If more than one device is attached, and more than one speed is specified, the devices will not necessarily all be of the same speed.

Resource Disposition

The mtape_ I/O module utilizes two types of resources: devices (tape drives) and volumes. Once an I/O switch is attached, resources are assigned to the user's process on a demand basis. When the I/O switch is detached, the default resource disposition unassigns all devices and volumes.

Write Rings And Write Protection

Before a volume can be written on, a write ring (an actual plastic ring) must be manually inserted into the reel. This can only be done before the volume is mounted on a device. When a volume is needed, the I/O module sends the operator a mount message that specifies if the volume is to be mounted with or without a ring.

In general, the decision of whether write rings are to be installed or not is made at attach time. This decision is effected by either the explicate use of the "-ring" attach description argument, or the current default value of the ring specification (See "Default Values" below). If output operations are to be performed on the volume set, then installation of write rings must be specified or an error will result when attempting to open a file for output or input_output. The write ring decision may be effected after the attach is complete by the use of the "ring_in" control operation described below.

When a volume set is mounted with write rings and the I/O switch is opened for input, the hardware file protect feature is used to safeguard the file set. Conversely, when a volume set is mounted with write rings and has subsequently been opened for input and closed, if it is now to be opened for output or input_output, the hardware file permit feature is used to once again allow writing operations.

Error Processing

If an error occurs while reading, the I/O module makes 25 attempts to backspace and re-read, using the available hardware error recovery mechanisms. If an error occurs while writing, the I/O module makes 10 attempts to backspace, erase, and rewrite. If an unrecoverable error occurs while writing file labels or tapemarks, the user is queried as to preserving the defective file versus file set consistency. (See "Queries" below.) If an unrecoverable error occurs during certain phases of volume switching or label reading, the I/O switch may be closed. The overriding concern of the error recovery strategy is:

1. to maintain a consistent file set structure
2. to ensure the validity of data read or written

Opening

Opening is made through the iox_$open_file entry which supports a character string "open description" argument for supplying file specific attributes to the per-format modules (See "Open Description" below). The iox_$open entry is supported in the sense that it will forward the call to the mtape_$open_file entry, supplying a minimal open description by default. This default open description is different for each per-format module, refer to the section titled "Per-format Modules" below, for details.

With one exception, the mtape_ I/O module and its subordinate Per-Format modules have a record oriented interface and support sequential_input, sequential_output, and sequential_input_output opening modes. The exception is the Multics Per-Format module, which has a stream oriented interface and supports stream_input and stream_output opening modes only.

An I/O switch can be opened and closed any number of times in the course of a single attachment. All openings are governed by the same attach description.

Open Description

The open description is an ASCII character string argument to the iox_$open_file entry and provides a means of specifying the attributes of the desired file to be processed.

For input operations on one of the supported volume types, a null open description may be specified since all file attributes may be obtained from the file header label records or from default values (See Default Values below). For output operations, all attributes of a file must be specified either in the open description or by using their corresponding default values.

For readability, the first specification in the open description may be optionally non-hyphenated, followed by as many or as few hyphenated specifications as are necessary to define the desired operations on the specified file.

Only those open description specifications that are generic to all (or most all) of the supported standard labeled volume types are defined below. For open description specifications that are particular to a given type of labeled volume type, see their definition in the section titled "Per-Format Modules" below.

In general, the open description has the following form:

open_spec*1* open_spec*2* ..... open_spec*n*

where:

1. open_spec*1* open_spec*2* and open_spec*n*
   are a sequence of file specific attributes and may be chosen from the following:

   -block *b*, -bk *b*
   specifies the block length in characters, where the value of *b* may be dependent upon the value of *r* specified in the -record control argument.

   -comment STR, -com STR
   specifies a user comment to be displayed on the user_output I/O switch, after the file has been successfully opened. STR could be an informative message providing a visual check point to the user, when processing several files of a multifile volume set. For example, the comments "Begin processing the student master file" or "Begin payroll run", might be typical. If STR contains embedded white space (i.e. spaces or horizontal tabs), then it must be enclosed in quotes.

   -display, -ds
   specifies that the entire open description, after it

has been parsed and any necessary defaults added, will be displayed on the user_output I/O switch.

-expires date, -exp date
specifies the expiration date of the file to be created or generated, where date must be of a form acceptable to the convert_date_to_binary_ subroutine which is described in the MPM Subroutines.

-extend, -ext
specifies extension of an existing file.

-force, -fc
specifies that the expiration date of the file being overwritten is to be ignored.

-format *f*, -fmt *f*
specifies the record format, where *f* is a format code.

-last_file, -lf
specifies the file to be processed as the "last" file of the volume set.

-mode STR, -md STR
specifies the encoding mode used to record the file data, where STR is the string ascii, ebcdic, or binary.

-name STR, -nm STR
specifies the file identifier of the file where STR is from 1 to 17 characters.

-next_file, -nf
specifies the file to be processed as the "next" (or first) file on the volume set. For output operation, if -number and or -name are not specified, the values for their respective fields (if any, for the volume label standard being used), are fabricated as follows:

The file sequence number is set to the last file sequence number plus 1.
The file identifier is set to a character string representation of the file sequence number (i.e. FILE1, FILE99, etc.). If this fabricated file identifier has an identical character representation as a previous file identifier in the file set, then an iteration suffix is

appended to the new file identifier (i.e. FILE1.1, FILE99.1, etc.).

The -next_file argument is ignored if a -number or a -name argument are also specified in the open description. If an open description does not contain either a -name, -number or -next_file argument and if a previous close_file operation did not specify -beginning_of_file in the close description, then a -next_file argument is inserted by default.

-number N, -nb N
specifies the file sequence number, the position of the file within the file set, where N is an integer in the range $1 \leq N \leq 9999$.

-record r, -rec r
specifies the record length in characters, where the value of r may be dependent upon the choice of record format. (See "Creating a File" below.)

Close Operation

The I/O switch must be open. Closing is made through the iox_$close_file entry which supports a character string "close description" argument for supplying file specific attributes to the per-format modules (See "Close Description" below). The iox_$close entry is supported in the sense that it will forward the call to the mtape_$close_file entry, supplying a null close description.

Close Description

The close description is an ASCII character string argument to the iox_$close_file entry and provides a means of specifying actions to be taken when closing the current file.

For readability, the first specification in the close description may be optionally non-hyphenated, followed by as many or as few hyphenated specifications as are necessary to define the desired operations on the specified file.

Only those close description specifications that are generic to all (or most all) of the supported standard labeled volume types are defined below. For close description specifications that are particular to a given type of labeled volume type, see their definition in the section titled "Per-Format Modules" below.

In general, the close description has the following form:

close_spec1 close_spec2 ..... close_specn

where:

1. close_spec1 close_spec2 and close_specn
   are a sequence of attributes pertinent to the closing of the current file and may be chosen from the following:

-beginning_of_file, -bof
   specifies that the tape volume is to be positioned at the beginning of the current file, upon closing.

-comment STR, -com STR
   specifies a user comment to be displayed on the user_output I/O switch, after the file has been successfully closed. STR could be an informative message providing a visual check point to the user, when processing several files of a multifile volume set. For example, the comments "Completed processing the student master file" or "End payroll run", might be typical. If STR contains embedded white space (i.e. spaces or horizontal tabs), then it must be enclosed in quotes.

-display, -ds
   specifies that the entire close description, after it has been parsed and any necessary defaults added, will be displayed on the user_output I/O switch.

-end_of_file, -eof
   specifies that the tape volume is to be positioned at the end of the current file upon closing.

-leave
   specifies that the tape volume is to remain at its current position, upon closing.

Note:
   The -beginning_of_file, -end_of_file and the -leave control arguments are mutually exclusive. If more that one of these control arguments appear in the close description, then the last one will take

precedence. If none of these control control arguments are specified, then the -leave control argument is inserted by default.

Control Operation

The mtape_ I/O module supports a variety of control operations.

change_module
file_set_status, fsst
file_status
force_end_of_volume, feov
hardware_status, hws
ring_in
volume_set_status, vsst
volume_status, vst

In the descriptions below, info_ptr is the information pointer specified in an iox_$control entry point call.

change_module OPERATION

This operation allows a user to switch to a different per-format module to process some piece of a particular tape volume if he so desires. The I/O switch must be closed. A typical use for this control order is to switch from one of the other per-format modules to the "raw" per-format module to perform some raw operations. The change_module operation also allows a user to specify his own per-format module that doesn't happen to be named one of the standard names (i.e. multics, ansi, ibm, gcos, cp6, raw, or unlabeled, followed by the string "_tape_io_"). The info_ptr points to a char (*) varying string which indicates what per-format module the user wishes to use (i.e. For the standard per-format modules, this character string would be "multics", "ansi", "ibm", "gcos", "cp6", "unlabeled", or "raw"). A search is then made, using the search_path mechanism, for this string with "_tape_io_" appended to it for the desired module. If the info_ptr is null, then this is an indication that the user wishes to "pop" back to the original per-format module, which is allowed if the user is open for input. In that case the current tape volume is repositioned to a known position by rewinding before control is given back to the original per-format module. If the info_ptr was null but the "change_module" operation has never been called and there is no other module to "pop" back to, then the change_module control operation is ignored. If the user performs any output type operations while he is executing in the new per-format module, the request to "pop" back to the original per-format module is rejected with an error.

file_set_status OPERATION

This operation may be used to obtain information about the entire file set as opposed to just the current file. The info_ptr should point to an extendable area which the mtape_ I/O module will fill with a structure of the following form:

```
     dcl 1 fsst aligned based (info_ptr),
         2 fsst_type fixed bin,
         2 nfiles fixed bin,
         2 fs_status (0 refer (fsst.nfiles)),
           3 file_state fixed bin,
           3 error_code fixed bin (35),
           3 file_id char (32),
           3 begin_vol_index fixed bin,
           3 end_vol_index fixed bin,
           3 file_sections fixed bin,
           3 generation fixed bin,
           3 gen_version fixed bin,
           3 creation char (5),
           3 expiration char (5),
           3 file_format fixed bin,
           3 blklen fixed bin,
           3 reclen fixed bin (21),
           3 mode fixed bin,
           3 blkcnt fixed bin (35);
```

where:

1. fsst_type
   is the same as the label_type field defined in the volume_status operation defined below.

2. nfiles
   is the number of files in the file set.

3. fs_status
   is an array of structures of file set members, which appears below in sequential order.

4. file_state
   is the current state of this file and could have one of the following values:

   0 = No information available (I/O switch never opened)
   1 = File not open
   2 = File open
   3 = File open and locked for error

The "locked for error" state referenced above is defined as an error or circumstance that prevents continued processing of this file. For example, parity error while reading, reached end of information, no next volume available, etc.

5. error_code
   is the error code when file_status.state = 3 above, otherwise equal to 0.

6. file_id
   is the file name or identifier as recorded in the appropriate file label record. This field will be blank for those formats that have no file identifier field.

7. begin_vol_index
   is an index to the first volume set member on which this file resides.

8. end_vol_index
   is an index to the last volume set member on which this file resides.

9. file_sections
   is a count of the number of volumes on which this file resides.

10. generation
    is the generation number of this file for those formats that support several "generations" of files. If this is the first generation, or if the format does not support several generations, then this field will be equal to 0.

11. gen_version
    is the generation version number for those formats that supports file generations. If this is the first generation, or if the format does not support several generations, then this field will be equal to 0.

12. creation
    is the Julian creation date of this file.

13. expiration
    is the Julian expiration date of this file.

14. file_format
    is the numeric value of the current file format. Although this is per-format module specific, the

following generic values will be recognized by all per-format modules:

0 = not specified
1 = FB (fixed length blocked)
2 = DB or VB (variable length blocked)
3 = S (spanned)
4 = SB (spanned blocked)

15. blklen

is the maximum block length of each block within this file.

16. reclen

is the maximum record length of logical records within this file.

17. mode

is a numeric indication of the recorded mode of this file. The following values are acceptable:

1 = ASCII
2 = EBCDIC
3 = Binary
4 = BCD

18. blkcnt

is the number of tape blocks contained in this file. If this file is still open for input or output, this number represents the number of blocks processed thus far.

hardware_status OPERATION

This operation returns a structure that contains the raw IOM status and the english language description of this status, generated by the last tape I/O operation. The I/O switch must be open. The structure to which info_ptr points, is declared as follows:

```
dcl 1 hardware_status  based (info_ptr) aligned,
      2 IOM_bits bit (72),
      2 description char (256) varying;
```

where:

1. IOM_bits

is the raw IOM hardware status.

2. description
   is the English language description of this hardware status.

ring_in OPERATION

This operation will cause subsequent volume mounts to be requested with write rings installed. The I/O switch must be closed and the info_ptr set to null. The effect of this operation is to cause the current volume to be demounted and the write ring indicator to be set in the internal data base maintained by mtape_. At the time of the next file opening, the appropriate volume will be requested to be mounted with a write ring installed. If write rings have already been requested to be installed, either by the use of the "-ring" attach description argument, or by a previous invocation of the ring_in control operation, then the ring_in control operation is considered a "no-op" and has no effect.

volume_status OPERATION

This operation returns a structure that contains the status of the current volume. If the I/O switch is open, the current volume is the volume on which the file section currently being processed resides. If the switch has never been opened, the current volume is the first (or only) volume in the volume set. If the switch was opened, but is now closed, the current volume is that on which the last file section processed resides. The structure to which info_ptr points, is declared as follows:

```
dcl 1 volume_status based (info_ptr) aligned,
      2 volume_name char (32),
      2 volume_id char (32),
      2 label_type fixed bin,
      2 volume_seq fixed bin,
      2 device_name char (8),
      2 read_errors fixed bin (35),
      2 write_errors fixed bin (35);
```

where:

1. volume_name
   is the name of the current volume as specified in the volume sequence list (i.e. attach description).

2. volume_id

is the name of the current volume as recorded in the volume label. For unlabeled volumes, this field will be blank.

3. label_type
   is the label type of this volume and could have one of the following values:

   0 = unlabeled
   1 = ANSI
   2 = IBM
   3 = Multics
   4 = CP6
   5 = GCOS FRC
   6 = GCOS UFAS

4. volume_seq
   is the order of this volume within the volume set.

5. device_name
   is the name of the tape device that this volume is mounted on (e.g. "tape_01"). If the volume is currently unmounted, this field will be blank.

6. read_errors
   is a count of the current number of read errors that have occurred on this tape volume.

7. write_errors
   is a count of the current number of write errors that have occurred on this tape volume.

file_status OPERATION

This operation returns a structure that contains the current status of the file specified in the open description. If the I/O switch has never been opened, no information can be returned; this situation is indicated by file_status.file_state = 0. If the switch was opened, but is now closed, the current status of the file is its status just prior to closing. The structure to which info_ptr points, is declared as follows:

```
dcl 1 file_status based (info_ptr) aligned,
    2 file_state fixed bin,
    2 error_code fixed bin (35),
    2 label_type fixed bin,
    2 file_id char (32),
    2 file_seq fixed bin,
    2 begin_vol_index fixed bin,
    2 end_vol_index fixed bin,
    2 file_sections fixed bin,
    2 generation fixed bin,
    2 gen_version fixed bin,
    2 creation char (5),
    2 expiration char (5),
    2 file_format fixed bin,
    2 blklen fixed bin,
    2 reclen fixed bin (21),
    2 mode fixed bin,
    2 blkcnt fixed bin (35);
```

where:

1. file_state
   is the current state of this file and could have one of the following values:

   0 = No information available (I/O switch never opened)
   1 = File not open
   2 = File open
   3 = File open and locked for error

   The "locked for error" state referenced above is defined as an error or circumstance that prevents continued processing of this file. For example, parity error while reading, reached end of information, no next volume available, etc.

2. error_code
   is the error code when file_status.state = 3 above, otherwise equal to 0.

3. label_type
   is the same as the label_type field defined in the volume_status operation defined above.

4. file_id
   is the file name or identifier as recorded in the appropriate file label record. This field will be blank for those formats that have no file identifier field.

5. file_seq
   is the order of this file within the file set.

6. begin_vol_index
   is an index to the first volume set member on which this file resides.

7. end_vol_index
   is an index to the last volume set member on which this file resides.

8. file_sections
   is a count of the number of volumes on which this file resides.

9. generation
   is the generation number of this file for those formats that support several "generations" of files. If this is the first generation, or if the format does not support several generations, then this field will be equal to 0.

10. gen_version
    is the generation version number for those formats that supports file generations. If this is the first generation, or if the format does not support several generations, then this field will be equal to 0.

11. creation
    is the Julian creation date of this file

12. expiration
    is the Julian expiration date of this file.

13. file_format
    is the numeric value of the current file format. Although this is per-format module specific, the following generic values will be recognized by all per-format modules:

    0 = not specified
    1 = FB (fixed length blocked)
    2 = DB or VB (variable length blocked)
    3 = S (spanned)
    4 = SB (spanned blocked)

14. blklen
    is the maximum block length of each block within this file.

15. reclen
    is the maximum record length of logical records within this file.

16. mode
    is a numeric indication of the recorded mode of this file. The following values are acceptable:

1 = ASCII
2 = EBCDIC
3 = Binary
4 = BCD

17. blkcnt

is the number of tape blocks contained in this file. If this file is still open for input or output, this number represents the number of blocks processed thus far.

feov OPERATION

This operation forces the end of a volume and initiates volume switching when writing a file. The switch must be open for output. The operation is equivalent to detection of the end of tape reflective strip. The info_ptr should be a null pointer.

volume_set_status OPERATION

This operation may be used to obtain information about the entire volume set as opposed to just the current volume. The info_ptr should point to an extendable area which the mtape_ I/O module will fill with a structure of the following form:

```
     dcl 1 vsst aligned based (info_ptr),
           2 vsst_type fixed bin,
           2 nvols fixed bin,
           2 vs_status (0 refer (vsst.nvols)),
             3 volume_name char (32),
             3 volume_id char (32),
             3 mounted bit (1),
             3 device_name char (8),
             3 read_errors fixed bin (35),
             3 write_errors fixed bin (35);
```

where:

1. vsst_type
   is the label type of this volume set and could have one of the following values:

   0 = unlabeled
   1 = ANSI
   2 = IBM
   3 = Multics
   4 = CP6
   5 = GCOS FRC
   6 = GCOS UFAS

2. nvols
   is the number of volumes in the volume set.

3. vs_status
   is an array of structures of volume set members, which appears below in sequential order.

4. volume_name
   is the name of this volume set member as specified in the volume sequence list (i.e. attach description).

5. volume_id
   is the name of this volume set member as recorded in the volume label. For unlabeled volumes, this field will be blank.

6. device_name

is the name of the tape device that this volume set member is currently mounted on (e.g. "tape_01"). If the volume is currently unmounted, this field will be blank.

7. read_errors
   is a count of the current number of read errors that have occurred on this tape volume.

8. write_errors
   is a count of the current number of write errors that have occurred on this tape volume.

Detach Operation

The I/O switch must be closed. Detachment is made through the iox_$detach entry which supports a character string "detach description" argument for supplying volume-set specific information for the disposition of the volume-set (See "Detach Description" below). The iox_$detach_iocb entry is supported in the sense that it will forward the call to the mtape_$detach entry, supplying a null detach description.

If the I/O module determines that the membership of the volume set might have changed, the volume set members are listed before the set is demounted; volumes not listed are available for incorporation into other volume sets.

Detach Description

The detach description is an ASCII character string argument to the iox_$detach entry and provides a means of specifying actions to be taken when detaching the current volume set.

For readability, the first specification in the detach description may be optionally non-hyphenated, followed by as many or as few hyphenated specifications as are necessary to define the desired operations on the specified file.

Only those detach description specifications that are generic to all (or most all) of the supported standard labeled volume types are defined below. For detach description specifications that are particular to a given type of labeled volume type, see their definition in the section titled "Per-Format Modules" below.

In general, the detach description has the following form:

detach_spec1 detach_spec2 ..... detach_specn

where:

1. detach_spec1 detach_spec2 and detach_specn
   are a sequence of attributes pertinent to the detachment of the current volume set and may be chosen from the following:

   -comment STR, -com STR
   allows the optional specification of a message to be displayed on the operators console at the time the volume set is to be detached. The comment text, STR, may be from 1 to 64 characters in length and must be quoted if it contains embedded white space.

   -display, -ds
   specifies that the entire detach description, after it has been parsed and any necessary defaults added, will be displayed on the user_output I/O switch.

   -unload
   specifies that any members of the volume set currently mounted are to be demounted at the time of detachment.

   -rewind
   specifies that any members of the volume set currently mounted are to be rewound to load point at the time of detachment. This is the default in the absence of the -unload control argument.

Modes Operation

The mtape_ I/O module does not support the modes operation.

Position Operation

In general, the mtape_ I/O module supports all positioning modes when the I/O switch is open for input or input_output. Some restrictions apply to the individual per-format modules. See the section entitled "Per-Format Modules" for details.

Read Length Operation

The I/O switch must be open for sequential_input, or sequential_input_output.

Read Record Operation

The I/O switch must be open for sequential_input, or sequential_input_output.

Write Record Operation

The I/O switch must be open for sequential_output, or sequential_input_output. Unlike previous tape I/O modules, non-mod 4 byte records may be written.

Get Chars Operation

The I/O switch must be open for stream_input, or stream_input_output.

Put Chars Operation

The I/O switch must be open for stream_output, or stream_input_output.

Control Operations from Command Level

All control operations supported by this I/O module can be executed from command level by using the io_call command. The general format is:

```
io_call control switchname operation -control_arg where:
```

1. switchname
   is the name of the I/O switch that is attached through the I/O module to an ANSI tape file-set.

2. operation
   is any of the control operations previously described.

3. control_arg
   is an operation control argument.

Queries

Under certain exceptional circumstances, the I/O module queries the user for information needed for processing to continue or instructions on how to proceed.

Querying is performed by the command_query_ subroutine. The user may intercept one or more types of query by establishing a handler for the command_question condition, that is signalled by the command_query_ subroutine. Alternately, the answer command (described in the MPM Commands) can be used to intercept all queries. The use of a predetermined "yes" answer to any query causes those actions to be performed that attempt to complete an I/O operation without human intervention.

In the following list of queries, status_code refers to command_question_info.status_code. See the MPM Reference Guide for information regarding the command_question condition and the command_question_info structure.

status_code = error_table_$file_aborted

This can occur only when the I/O switch is open for output. The I/O module is unable to correctly write file header labels, trailer labels, or tapemarks. This type of error invalidates the structure of the entire file set. Valid file set structure can only be restored by deleting the defective file or file section from the file set.

The user is queried for permission to delete the defective file or file section. If the response is "yes", the I/O module attempts deletion. The attempt may or may not succeed; the user is informed if the attempt fails. If the response is "no", no action is taken. The user will probably be unable to subsequently process the file, or append files to the file set; however, this choice permits retrieval of the defective file with another I/O module. In either case, the I/O switch is closed.

status_code = error_table_$unexpired_volume

This can occur only when the I/O switch is open for output. A volume must be either reinitialized or overwritten; however, the first file or file section on the volume is unexpired.

The user is queried for permission to initialize or overwrite the unexpired volume. If the response is "yes", the volume is initialized or overwritten and processing continues. If the response is "no", further processing cannot continue, and the I/O switch is closed.

status_code = error_table_$uninitialized_volume

A volume requires reinitialization or user verification before it can be used to perform any I/O. The I/O module distinguishes among four causes by setting command_question_info.query_code as follows:

query_code = 1	the first block of the tape is unreadable. The tape is either defective, or recorded at an invalid density. This query code can occur only if the I/O stream is opened for output.
query_code = 2	the first block of the tape is not a valid volume label for the volume type specified in the "-label" attach description control argument. This query code can occur only if the I/O stream is opened for output.
query_code = 3	the volume identifier recorded in the volume label is incorrect. The volume identifier does not match the volume name.
query_code = 4	the density at which the volume is recorded is incorrect. The volume density does not match the specified density. This query code can occur only if the I/O stream is opened for output.

If the I/O switch is opened for output, the user will be asked whether he wants to initialize or re-initialize the volume. If the I/O switch is opened for input, the user will be asked whether he wants to continue processing in spite of the discrepancy. If the response is "yes", the volume is reinitialized and processing continues. If the response is "no", further processing cannot continue, and the I/O switch is closed.

status_code = error_table_$unexpired_file

This can occur only when the I/O switch is open for output. A file that must be extended, modified, generated, or replaced is unexpired.

The user is queried for permission to overwrite the unexpired file. If the response is "yes", processing continues. If the response is "no", further processing cannot continue, and the I/O switch is closed.

status_code = error_table_$no_next_volume

This can occur when reading a multivolume file, or when writing a file and reaching physical end of tape. The I/O module is unable to determine the name of the next volume in the volume set.

The user is queried for permission to terminate processing. If the response is "yes", no further processing is possible. If the I/O switch is open for output, the I/O switch is closed. If the response is "no", the user is queried for the volume name of the next volume. (See status_code = 0 below.)

status_code = 0

This occurs only when the response to the above query is "no". The user is requested to supply the name of the next volume. The response may be a volume name, optionally followed by a mount message. Even if the volume name begins with a hyphen, it must not be preceded by the -volume control argument. If a mount message is to be specified, the response takes the following form:

```
volume_name -comment STR
```

where STR is the mount message and need not be a contiguous string. See "Volume Specification" above. This is the only query that does not require a "yes" or "no" response. If a preset "yes" is supplied to all queries, this particular query never occurs.

Default Values

As an ease of use feature, all control arguments and their associated values that a user may specify in the attach, open, close and detach descriptions, is supplied with a reasonable default value by mtape_ and or the per-format module currently in execution. There are two classes of defaults contained within mtape_ and its associated per-format modules:

Global defaults
default values that pertain to all formated tape types.

Per-Format defaults
default values that differ (or may differ) for each per-format module.

All default values are created at first reference in the users default value segment (normally located at [home_dir]>[user name].value). The global default values are created by mtape_ proper and the per-format defaults are created by each per-format module, during its initialization sequence. After their initial creation, the default values can be changed and manipulated by the user, using the value_set command.

Each default value has a three component name, with the global defaults being in the form of "mtape_.global.<value_name>" and the per-format defaults being in the form "mtape_.<vol_type>.<value_name>". The values themselves are stored as an ASCII character string. Numeric values are converted when used by mtape_, and bit string switches are stored as "true" or "false". Listed below are the global defaults, with the default name, its initial value and other possible values. The per-format defaults will be listed in the documentation of each per-format module.

Default Name	Initial Value	Other Possible Values
mtape_.global.density	1600	800, 6250, 200, 556
mtape_.global.label	ansi	ANSI, ibm, IBM, gcos, GCOS, multics, Multics, cp6, CP6, raw, RAW
mtape_.global.no_labels	false	true
mtape_.global.ring	false	true
mtape_.global.tracks	9	7
mtape_.global.device	1	2, 3, 4, ... 63
mtape_.global.speed	0 (any)	75, 125, 200

Per-Format Modules

In order to process a variety of different tape volume formats, the mtape_ I/O module employs standard subroutine interfaces to what are known as Per-Format modules. The generic name of each of these Per-Format modules or subroutines is <vol_type>_tape_io_, where <vol_type> represents the identified name of the volume format which is to be processed. Seven Per-Format modules are currently planned in support of mtape_. They are:

ansi_tape_io_	For ANSI standard tapes
ibm_tape_io_	For IBM standard tapes
multics_tape_io_	For Multics standard tapes
gcos_tape_io_	For GCOS standard tapes
cp6_tape_io_	For CP6/CP5 standard tapes
unlabeled_tape_io_	For unlabeled or unrecognized format tapes
raw_tape_io_	For processing tapes in a "raw" or user controlled fashion

The Per-Format modules are externally callable and are found in the storage system via the search_path mechanism. For tape input, RCP returns the volume type of the tape volume just mounted, as one of the <vol_types> mentioned above (except for the raw per-format module which must always be explicitly requested with the "-label raw" attach description argument). For tape output, the volume type is determined from either the attach description "-label" specification or by the appropriate default value of same. After the volume type has been determined by this procedure, mtape_ searches for the appropriate module in the search paths.

From the above discussion, it should be easy to see that a user could substitute his own versions of these standard modules by first writing his own subroutines and then changing the search paths so that his version would be found before the standard system version.

Per-Format Module Selection

Selection of the appropriate Per-Format module to process the desired volume set is performed at attach time. Information returned by RCP after a successful volume mount, as well as the presence of the "-label" and "-no_labels" attach description arguments or their current default values, are all used in the Per-Format module selection process. However, there is no knowledge available at attach time that specifies whether the user will open for tape input or output. Even the presence of the "-ring" attach description argument is no guarantee that the user will open for output operations. The Per-Format module then, may have to take some action upon opening to fulfill requirements of special situations. The table below, and the annotated comments that follow it, illustrate the relationship between the Per-Format module selection process and any special action that must be taken at open time, by the selected Per-Format module.

Volume type returned by RCP	-lbl & -nlbl attach description values	Per-Format module selected	Special action upon opening	
			input	output
ansi	none	ansi_tape_io_		
ibm	none	ibm_tape_io_		
gcos	none	gcos_tape_io_		
multics	none	multics_tape_io_		(1)
multics	-lbl multics	multics_tape_io_		(1)
cp6	none	cp6_tape_io_		
unlabeled	none	unlabeled_tape_io_		N/A
unlabeled	none	ansi_tape_io_ (2)	N/A	
unlabeled	-nlbl	unlabeled_tape_io_		
unreadable	none	ansi_tape_io_ (2)	(3)	
ansi	-lbl raw	raw_tape_io_		(4)
unlabeled	-lbl ibm	ibm_tape_io_	(5)	(6)
unlabeled	-lbl ibm -nlbl	ibm_tape_io_		
ibm	-lbl ibm -nlbl	ibm_tape_io_	(7)	(4)
unlabeled	-lbl gcos	gcos_tape_io_		(6)
unlabeled	-lbl gcos -nlbl	gcos_tape_io_		
ansi	-lbl ibm	ibm_tape_io_	(8)	(4)
ibm	-lbl gcos -nlbl	gcos_tape_io_	(8)	(4)
unlabeled	-lbl cdc	cdc_tape_io_ (9)		
unlabeled	-lbl dec	dec_tape_io_ (9)		
unreadable	-lbl ibm	ibm_tape_io_	(3)	
unreadable	-lbl raw	raw_tape_io_		
any readable tape vol but not at req. dens	matching label type or none	<lab_type>_tape_io_	(10)	(11)

(1) The volume label is re-written each time it is opened for output.

(2) This is really the value of the global default, mtape_.global.label and not necessarily ansi. This default value is set (created at first reference) by mtape_ but may be changed by the user at any time. Note that in the case of an unlabeled tape being detected by RCP and no "-label" (-lbl) or "-no_labels" (-nlbl) specification, the unlabeled Per-Format module is selected for input operations while the default Per-Format module is selected for output operations. This would seem to break the rule that no knowledge of opening mode is known at the time the Per-Format module is selected. In actuality, the unlabeled Per-Format module is selected at attach time. However, when the open operation is executed and the opening mode is known, a special feature of the unlabeled Per-Format module is invoked when it is

determined to be an output mode. This special feature determines what the current default Per-Format module is and does the equivalent of a "change_module" operation to call the current default Per-Format module into execution.

(3) Although a "-label" (-lbl) specification was given, the Per-Format module will abort during its volume initialization sequence. Only the "RAW" (or a user defined) Per-Format module is allowed to process a tape volume for input, whose label is deemed unreadable.

(4) The user is queried before allowing the destruction, or potential destruction of any labeled volume set.

(5) The volume is processed as unlabeled.

(6) Volume is initialized with standard labels.

(7) Since RCP determined that this is a standard labeled volume, the "-no_labels" (-nlbl) specification is ignored.

(8) This is considered an error because the attach description label specification and the actual volume label determined by RCP, do not agree as to their type, causing an inconsistency to exist. The opening is aborted.

(9) These are examples of the use of user defined Per-Format modules. Note that the value used in the "-label" (-lbl) attach description specification, has the string "_tape_io_" appended to it to complete the Per-Format module name that is searched for.

(10) Input operations will proceed at the density returned by RCP, and any density specification requested by the user is ignored.

(11) In general, the tape volume (including the volume label(s)) will be re-written at the user requested density. If it is determined that the tape unit on which the tape volume is currently mounted does not have the capability of writing at the user requested density, then the user is queried if he wants to write at the RCP determined density, or have the tape volume re-mounted on a different tape unit and initiate output operations at the user requested density.

Per-Format Module Interface

In order to provide adequate processing capabilities for each of the Per-Format modules, seven (7) standard entry points have been defined. They are:

```
<vol_type>_tape_io_$volume_open
<vol_type>_tape_io_$volume_close
<vol_type>_tape_io_$file_open
<vol_type>_tape_io_$file_close
<vol_type>_tape_io_$read
<vol_type>_tape_io_$write
<vol_type>_tape_io_$order
```

Below is a discussion of each of these entries and the general function of each:

<vol_type>_tape_io_$volume_open entry

The task of this entry is to process the volume label (or labels) and do any house keeping functions that may be required by the individual per-process modules (e.g. Fill in pertinent information in the "volume info structure" either from the volume label on input or from the attach description on output. On input, compare the recorded volume name to the volume sequence list and check for discrepancies, read and save any user volume labels for later requests by the user to "see" these label records. On output, write the standard volume label sequence, etc.).

Usage

```
     dcl <vol_type>_tape_io_$volume_open entry
          (ptr, fixed bin (35));
     call <vol_type>_tape_io_$volume_open (vol_info_ptr, code);
```

where:

1. vol_info_ptr
is a pointer to the volume info structure defined below. (INPUT)

2. code
is a standard I/O system status code. (OUTPUT)

volume info structure

The volume info structure is the "root" of a tree structured linked list of volume set and file set information structures. It is allocated and initialized during attachment, with values for initialization coming from either the attach description, defaults or from RCP. It is updated with volume set and file set information as new volume set members are added to the volume sequence list and new files are created or recognized. The volume_info structure is deallocated (freed) at detach time.

```
     dcl 1 volume_info aligned based (vol_info_ptr),
           2 version fixed bin,
           2 label_type fixed bin,
           2 first_file_ptr ptr,
           2 last_file_ptr ptr,
           2 vs_head ptr,
           2 vs_tail ptr,
           2 vs_current ptr,
           2 density fixed bin,
           2 tracks fixed bin,
           2 speed fixed bin,
           2 mode fixed bin,
           2 ring bit (1),
           2 attach_desc_ptr ptr;
```

where:

1. version
   is the version number of this structure, currently 1.

2. label_type
   is the label type of this volume set and could have one of the following values:

   0 = unlabeled
   1 = ANSI
   2 = IBM
   3 = Multics
   4 = CP6
   5 = GCOS FRC
   6 = GCOS UFAS

3. first_file_ptr
   is a pointer to the first file info structure.

4. last_file_ptr
   is a pointer to the last file info structure.

5. vs_head
   is a pointer to the first volume_set member, defined below in the volume_set structure.

6. vs_tail
   is a pointer to the last volume_set member, defined below in the volume_set structure.

7. vs_current
   is a pointer to the current volume_set member defined below in the volume_set structure.

8. density
is a numeric indication of the volume_set density and has the following possible values:

0 = unspecified
1 = 800 BPI NRZI
2 = 1600 BPI PE
3 = 6250 BPI GCR
4 = 200 BPI NRZI
5 = 556 BPI NRZI

9. tracks
is an indication of the number of recorded tracks on each volume of the volume set and can have a value of 7 or 9 for 7 or 9 track tapes.

10. speed
is the tape unit speed to request that the volume set be mounted on and may have the following values:

```
0   = Unspecified, use any speed device
75  = Use a tape device whose speed is 75 IPS.
125 = Use a tape device whose speed is 125 IPS.
200 = Use a tape device whose speed is 200 IPS.
```

11. mode
specifies the hardware mode to be used in processing the volume set and has the following possible values:

0 = unspecified or variable modes
1 = binary
2 = nine
3 = bcd

12. ring
specifies if a write ring is to be installed in each volume of the volume set. A value of "0"b = no write ring and a value of "1"b = install write ring.

13. attach_desc_ptr
is a pointer to a copy of the original unparsed attach description.

<u>volume set structure</u>

Each volume of a volume sequence list has a volume_set structure associated with it. As each volume specification in the attach description is parsed, a volume_set structure is allocated and initialized for it. For a multi-volume volume set, these volume_set structures are chained together in a linked list and comprise the volume sequence list. Information in this structure is updated as each volume is mounted. If the volume set membership is increased as the result of a user query (See "Queries" above), then a new volume_set structure is allocated and initialized for each new volume added by the user. Each volume_set structure is deallocated (freed) at detach time.

```
     dcl 1 volume_set aligned based (vs_ptr),
         2 version fixed bin,
         2 mounted bit (1),
         2 ever_mounted bit (1),
         2 pad fixed bin,
         2 device_name char (8),
         2 prev_vs_ptr ptr,
         2 next_vs_ptr ptr,
         2 volume_name char (32),
         2 volume_id char (32),
         2 volume_comment char (64),
         2 first_vl_ptr ptr,
         2 last_vl_ptr ptr,
         2 first_uvl_ptr ptr,
         2 last_uvl_ptr ptr,
         2 read_errors fixed bin (35),
         2 write_errors fixed bin (35);
```

where:

1. version
   is the version number of this structure, currently 1.

2. mounted
   is an indication as to this volumes current mounted state. "0"b => not mounted, "1"b => mounted.

3. ever_mounted
   is an indication as to whether this volume has ever been mounted. "0"b => never mounted, "1"b => currently or was mounted.

4. device_name
   is the name of the tape device that this volume is mounted on. (e.g. "tape_01"). If the volume has never been mounted, this field will be padded with blanks. If the volume has been mounted, but is currently unmounted (indicated by the state switches, mounted = "0"b, ever_mounted = "1"b), this field will contain the device name of device last mounted on.

5. prev_vs_ptr

is a pointer to the previous volume_set members volume_set structure.

6. next_vs_ptr
   is a pointer to the next volume_set members volume_set structure.

7. volume_name
   is the volume name specified in the attach description for each volume set member.

8. volume_id
   is name of this volume set member as recorded in the volume label. For unlabeled volumes or volumes that have yet to be mounted, this field will be padded with blanks.

9. volume_comment
   is the attach time comment to be displayed on the operators console when the current volume is mounted, and may be blank.

10. first_vl_ptr
    is a pointer to the first volume label record structure, defined below by the label_record structure.

11. last_vl_ptr
    is a pointer to the last volume label structure, defined below by the label_record structure.

12. first_uvl_ptr
    is a pointer to the first user volume label record, if any and may be null.

13. last_uvl_ptr
    is a pointer to the last user volume label record, if any and may be null.

14. read_errors
    is a count of the current number of read errors that have occurred on this tape volume.

15. write_errors
    is a count of the current number of write errors that have occurred on this tape volume.

label record structure

All supported labeled volume types have associated with them volume label records and most have file label records as well. Each label record, as it is recognized as such, has a label record structure allocated for it as a member of a linked list of from 1 to N label record structures for each volume and or file. Storage for each label record structure (as well as for the contents of the label record), is allocated as each volume is mounted or each file is recognized respectively. Each label record structure is deallocated at detach time.

```
     dcl 1 label_record aligned based (lr_ptr),
         2 version fixed bin,
         2 mode fixed bin,
         2 prev_lab_ptr ptr,
         2 next_lab_ptr ptr,
         2 conversion fixed bin,
         2 lab_length fixed bin,
         2 lab_ptr ptr;
```

where:

1. version
   is the version number of this structure, currently 1.

2. mode
   is the hardware mode this label record is recorded in. Numerical values are assigned to mode as follows:

   1 = binary
   2 = nine
   3 = bcd

3. prev_lab_ptr
   is a pointer to the previous label record structure, if any.

4. next_lab_ptr
   is a pointer to the next label record structure if any.

5. conversion
   is a numeric indication of any character set conversion which must be done on the data being read from the tape or the data being written on the tape. The following values are acceptable:

   0 = no conversion
   1 = ASCII <--> EBCDIC

2 = ASCII <--> BCD

6. lab_length
   is the length of the actual label record data in 9 bit bytes.

7. lab_ptr
   is a pointer to the actual data in the label record which is volume format type specific.

<vol_type>_tape_io_$volume_close entry

The task of this entry is to close out processing of the current volume. This may mean writing the end of volume sequence on output, performing volume switching, etc.

Usage

```
     dcl <vol_type>_tape_io_$volume_close entry
          (ptr, fixed bin (35));
     call <vol_type>_tape_io_$volume_close (vol_info_ptr, code);
```

where:

1. vol_info_ptr
   is a pointer to the volume info structure defined above. (INPUT)
2. code
   is a standard I/O system status code. (OUTPUT)

<vol_type>_tape_io_$file_open entry

The task of this entry is to process the file label (or labels) and do any house keeping functions that may be required by the individual per-process modules (e.g. Fill in pertinent information in the "file info structure" either from the file label on input or from the open description on output. On input, read and save any user file labels for later requests by the user to "see" these label records. On output, write the standard file label sequence, etc.).

Usage

```
     dcl <vol_type>_tape_io_$file_open entry
          (ptr, fixed bin (35));
     call <vol_type>_tape_io_$file_open (file_info_ptr, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined below. (INPUT)
2. code
   is a standard I/O system status code. (OUTPUT)

file info structure

As each file is created on output or recognized on input, a file_info structure is allocated for it. Each file_info structure is chained together in a linked list so that information about this file is readily available for future reference. All file_info structures are deallocated (freed) at detach time.

```
dcl 1 file_info aligned based (file_info_ptr),
      2 version fixed bin,
      2 label_type fixed bin,
      2 vol_info_ptr ptr,
      2 prev_file_ptr ptr,
      2 next_file_ptr ptr,
      2 position,
        3 begin_vol_ptr ptr,
        3 end_vol_ptr ptr,
        3 cur_vol_ptr ptr,
        3 phy_file fixed bin (35),
        3 phy_block fixed bin (35),
        3 log_file fixed bin (35),
        3 log_record fixed bin (35),
        3 log_record_ptr ptr,
      2 file_format fixed bin,
      2 file_id char (32),
      2 seq_number fixed bin,
      2 block_size fixed bin (35),
      2 record_size fixed bin (35),
      2 char_size fixed bin,
      2 conversion fixed bin,
      2 open_mode fixed bin,
      2 first_file_lab_ptr ptr,
      2 last_file_lab_ptr ptr,
      2 first_uf_lab_ptr ptr,
      2 last_uf_lab_ptr ptr,
      2 first_file_trail_ptr ptr,
      2 last_file_trail_ptr ptr,
      2 first_uf_trail_ptr ptr,
      2 last_uf_trail_ptr ptr,
      2 open_desc_ptr ptr;
```

where:

1. version
   is the version number of this structure, currently 1.

2. label_type
   is the label_type duplicated from the volume_info structure above.

3. vol_info_ptr
   is a pointer to the volume_info structure for the volume set.

4. prev_file_ptr
   is a pointer to the previous files file_info structure.

5. next_file_ptr
   is a pointer to the next files file_info structure, and may be null.

6. position
   is a group of like information, indicating the current position of the file.

7. begin_vol_ptr
   is a pointer to the volume_set structure for the first volume on which this file resides (i.e. first file section).

8. end_vol_ptr
   is a pointer to the volume_set structure for the last volume on which this file resides (i.e. last file section).

9. cur_vol_ptr
   is a pointer to the volume_set structure for the current volume on which this file resides (i.e. current file section).

10. phy_file
    is the current physical file number within the current volume.

11. phy_block
    is the current physical block number within the current physical file.

12. log_file

is the number of the current logical file within the file set.

13. log_record

    is the number of the current logical record within the current block.

14. log_record_ptr

    is a pointer to the current or last logical record within the current block.

15. file_format

    is the numeric value of the current file format. Although this is per-format module specific, the following generic values will be recognized by all per-format modules:

    0 = not specified
    1 = FB (fixed block)
    2 = DB or VB (variable blocked)
    3 = S (spanned)
    4 = SB (spanned blocked)

16. file_id

    is the file identifier of the current file.

17. seq_number

    is the sequence number of the current file.

18. block_size

    is the maximum block size of all blocks in the current file.

19. record_size

    is the maximum record size of all records in the current file.

20. char_size

    is a numeric indicator of the number of bits in the characters of the datum of the current file. Values can be 1, for one bit, 6, for six bit characters, 9 for nine bit characters etc.

21. conversion

    is a numeric indication of any character set conversion which must be done on the data being read from the tape or the data being written on the tape. The following values are acceptable:

    0 = no conversion

1 = ASCII <--> EBCDIC
2 = ASCII <--> BCD

22. open_mode
is the numeric value of the iox_ mode (defined by the include file iox_modes.incl.pl1), for which this file is opened.

23. first_file_lab_ptr
is a pointer to the first file label record structure defined above.

24. last_file_lab_ptr
is a pointer to the last file label record structure.

25. first_uf_lab_ptr
is a pointer to the first user file label record structure, if any.

26. last_uf_lab_ptr
is a pointer to the last user file label record structure, if any.

27. first_file_trail_ptr
is a pointer to the first file trailer label record, defined by the label record structure above.

28. last_file_trail_ptr
is a pointer to the last file trailer label record, defined by the label record structure above.

29. first_uf_trail_ptr
is a pointer to the first user file trailer label record structure, if any.

30. last_uf_trail_ptr
is a pointer to the last user file trailer label record structure, if any.

<vol_type>_tape_io_$file_close entry

The task of this entry is to close out processing of the current file. This would mean writing the end of file trailer sequence on output, etc.

Usage

```
     dcl <vol_type>_tape_io_$file_close entry
```

```
          (ptr, fixed bin (35));
     call <vol_type>_tape_io_$file_close (file_info_ptr, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)
2. code
   is a standard I/O system status code. (OUTPUT)

<vol_type>_tape_io_$read entry

The task of this entry is to read logical records from a physical block and return the information to the user. This may include character set translation (i.e. EBCDIC to ASCII, BCD to ASCII), or format translation (i.e. expand compressed deck card images for gcos tapes, etc.)

Usage

```
     dcl <vol_type>_tape_io_$read entry
          (ptr, ptr, fixed bin (21), fixed bin (35));
     call <vol_type>_tape_io_$read
          (file_info_ptr, rcd_ptr, rcd_len, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)
2. rcd_ptr
   is a pointer to the logical record. (OUTPUT)
3. rcd_len
   is the logical record length. (OUTPUT)
4. code
   is a standard I/O system status code. (OUTPUT)

Note:
The read entry point is data demand driven. When the current tape block is exhausted, a common mtape_ subroutine entry point is called to obtain the next block. This common subroutine is defined below:

Usage

```
     dcl mtape_$read_block entry
```

```
        (ptr, ptr, fixed bin (21), fixed bin (35));
    call mtape_$read_block
        (file_info_ptr, block_ptr, block_length, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)

2. block_ptr
   is a pointer to a buffer containing the requested block. (OUTPUT)

3. block_length
   is the length of the block in 9 bit bytes. (OUTPUT)

4. code is a standard I/O system status code. (OUTPUT)

<vol_type>_tape_io_$write entry

The task of this entry is to write logical records to a physical block. This may include character set translation (i.e. ASCII to EBCDIC, ASCII to BCD), or format translation (i.e. compress source card images for gcos compressed deck formated tapes, etc).

Usage

```
     dcl <vol_type>_tape_io_$write entry
          (ptr, ptr, fixed bin (21), fixed bin (35));
     call <vol_type>_tape_io_$write
          (file_info_ptr, rcd_ptr, rcd_len, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)
2. rcd_ptr
   is a pointer to the logical record. (INPUT)
3. rcd_len
   is the logical record length. (INPUT)
4. code
   is a standard I/O system status code. (OUTPUT)

Note:

The write entry point is data demand driven. When the current tape block is full, a common mtape_ subroutine entry point is called to write out the block to tape. This common subroutine is defined below:

Usage

```
     dcl mtape_$write_block entry
         (ptr, ptr, fixed bin (21), fixed bin (35));
     call mtape_$write_block
         (file_info_ptr, block_ptr, block_length, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)

2. block_ptr
   is a pointer to a buffer containing the requested block. (INPUT)

3. block_length
   is the length of the block in 9 bit bytes. (INPUT)

4. code is a standard I/O system status code. (OUTPUT)

<vol_type>_tape_io_$order entry

The task of this entry is to process any control orders that are format specific and outside the realm of mtape_.

Usage

```
     dcl <vol_type>_tape_io_$order entry
          (ptr, char (*), ptr, fixed bin (35));
     call <vol_type>_tape_io_$order
          (file_info_ptr, order_name, info_ptr, code);
```

where:

1. file_info_ptr
   is a pointer to the file info structure defined above. (INPUT)
2. order_name
   is the name of the control order. Any control order not recognized by mtape_, will be passed on to the

per-format module. (INPUT)

3. info_ptr
is the information pointer present in the iox_$control call. The value of info_ptr may be null. (INPUT)

4. code
is a standard I/O system status code. (OUTPUT)