SERIES 60 (LEVEL 68)

MULTICS PROGRAMMERS' MANUAL
PERIPHERAL INPUT/OUTPUT

SUBJECT

Peripheral Input/Output Reference Material Including Command and I/O Module Descriptions

SPECIAL INSTRUCTIONS

This manual supersedes AX49, Rev. 0 dated June 1977 and its addendum (Addendum A dated January 1979).

SOFTWARE SUPPORTED

Multics Software Release 8.0

ORDER NUMBER

AX49-01

November 1979

Honeywell

PREFACE

This manual is one of six manuals that constitute the Multics Programmers' Manual (MPM). Primary reference material for user and subsystem programming on the Multics system is contained in these six manuals. Throughout this manual, references are frequently made to the MPM. For convenience, these references will be as follows:

Document	Referred To In Text As
Reference Guide (Order No. AG91)	MPM Reference Guide
Commands and Active Functions (Order No. AG92)	MPM Commands
Subroutines (Order No. AG93)	MPM Subroutines
Subsystem Writers' Guide (Order No. AK92)	MPM Subsystem Writers' Guide
Peripheral Input/Output (Order No. AX49)	MPM Peripheral I/O
Communications Input/Output (Order No. CC92)	MPM Communications I/O

The MPM Reference Guide contains general information about the Multics command and programming environments. It also defines items used throughout the rest of the MPM and, in addition, describes such subjects as the command language, the storage system, and the input/output system.

The MPM Commands is organized into four sections. Section 1 contains a list of the Multics command repertoire, arranged functionally. Section 2 describes the active functions. Section 3 contains descriptions of standard Multics commands, including the calling sequence and usage of each command. Section 4 describes the requests used to gain access to the system.

The MPM Subroutines is organized into three sections. Section 1 contains a list of the subroutine repertoire, arranged functionally. Section 2 contains descriptions of the standard Multics subroutines, including the declare statement, the calling sequence, and usage of each. Section 3 contains the descriptions of the I/O modules.

The MPM Subsystem Writers' Guide is a reference of interest to compiler writers and writers of sophisticated subsystems. It documents user-accessible modules that allow the user to bypass standard Multics facilities. The interfaces thus documented are a level deeper into the system than those required by the majority of users.

The MPM Peripheral I/O manual contains descriptions of commands and subroutines used to perform peripheral I/O. Included in this manual are commands and subroutines that manipulate tapes and disks as I/O devices.

The MPM Communications I/O manual contains information about the Multics Communication System. Included are sections on the commands, subroutines, and I/O modules used to manipulate communications I/O. Special purpose communications I/O, such as binary synchronous communication, is also included.

Throughout this manual, change bars in the margins indicate technical additions and changes; asterisks denote deletions.

Significant Changes in AX49, Addendum C

The description of administrative records in Section 3 has been expanded, and now includes a description of bootable tape label records.

The read_tape and query command has several new control arguments and requests described, and includes examples.

The tape_in and tape_out commands have a new maximum for tape file physical block length, as do the tape_ansi and tape_ibm I/O modules.

New control arguments and control orders have been documented for the tape_mult_ I/O module.

CONTENTS

ILLUSTRATIONS

TABLES

SECTION 1

INTRODUCTION

The Multics system supports input/output (I/O) operations on the following peripheral devices:

disk
magnetic tape
printer
card punch
card reader
communications lines

Although revisions of this manual will contain information on all of these devices, this manual presently documents I/O operations on magnetic tape, disk, and some forms of communications lines.

Section 2 describes two critical components of the Multics I/O facility: the resource control package (RCP) and the I/O interfacer (IOI). These descriptions are intended as reference information, since tape I/O can be performed by users with no knowledge of RCP and IOI.

Section 3 describes the standard Multics tape format used on tapes written and read by the tape_mult_ I/O module. Tape format required for the processing of tapes by the tape_ansi_ I/O module is described in the Draft Proposed Revision X3L5/419T of the American National Standards Institute's ANSI X3.27-1969, "Magnetic Tape Labels and File Structure for Information Interchange". Tape format required for the processing of tapes by the tape_ibm_ I/O module is described in the following IBM publications: OS Data Management Services Guide, Release 21.7, GC26-3746-2; IBM System 360 Disk Operating System Data Management Concepts, GC24-3427-8; and OS Tape Labels, Release 21, GC28-6680-4. Readers requiring information on ANSI and IBM tape formats should refer to these publications, since none of the material in them is duplicated in this manual.

Section 4 contains alphabetically arranged descriptions of I/O-related commands; Section 5 contains alphabetically arranged descriptions of the subroutines and I/O modules.

Section 6 gives some programming examples (in PL/I) illustrating the use of the user-ring peripheral I/O system.

No hardware status information is included in this manual. Readers desiring hardware status information should refer to the appropriate Multics Program Logic Manual.

The following commands handle various aspects of peripheral I/O:

assign_resource	list_resource_types
acquire_resource	print
cancel_resource	print_attach_table
close_file	print_request_types
console_output	reserve_resource
copy_file	set_cc
display_pl1io_error	set_tty
file_output	unassign_resource
io_call	vfile_adjust
line_length	vfile_status
list_resources	

All of these commands are described in the MPM Commands.

SECTION 2

PERIPHERAL I/O FACILITIES

Input/output in the user environment of the Multics system is organized around the user-ring I/O system subroutine, iox_. The entry points of iox_ provide for a general, device-independent interface supporting I/O and control functions. They may be called either via explicit PL/I code or via the facilities of language-provided I/O. Often, they are called internally from programs (see Section 4) that deal with peripheral I/O.

The user-ring I/O system is organized around I/O modules, programs that support the iox_ interfaces for a specific device, class of devices, or class of operations upon a given device or class of devices. (The available interfaces of iox_ are described in the MPM Subroutines.) I/O modules make appropriate calls upon the I/O interfaces of the supervisor, the resource control package, and the I/O interfacer to arrange for use of peripheral devices and perform operations upon them. The system provides a repertoire of I/O modules for peripheral devices. These I/O modules are documented in Section 5. The user may provide his own I/O modules as well. (See the MPM Subsystem Writers' Guide for guidelines for the implementation of I/O modules.)

The resource control package (RCP) is responsible for allocation and deallocation of peripheral devices to user processes. By means of RCP, user processes (and I/O modules) can gain access to peripheral devices. RCP provides for access checking and device selection. RCP is described in detail below.

The I/O interfacer (IOI) is the facility of the supervisor through which user programs (via I/O modules) can operate peripheral devices. IOI provides for the operation of the I/O hardware and the multiplexing of channels and other physical resources between processes. IOI can only be used to manipulate a device once a process has acquired the right to use that device via RCP. IOI is described below.

The interrelationship between user code, iox_, RCP, IOI, and the I/O modules is shown in Figure 2-1.

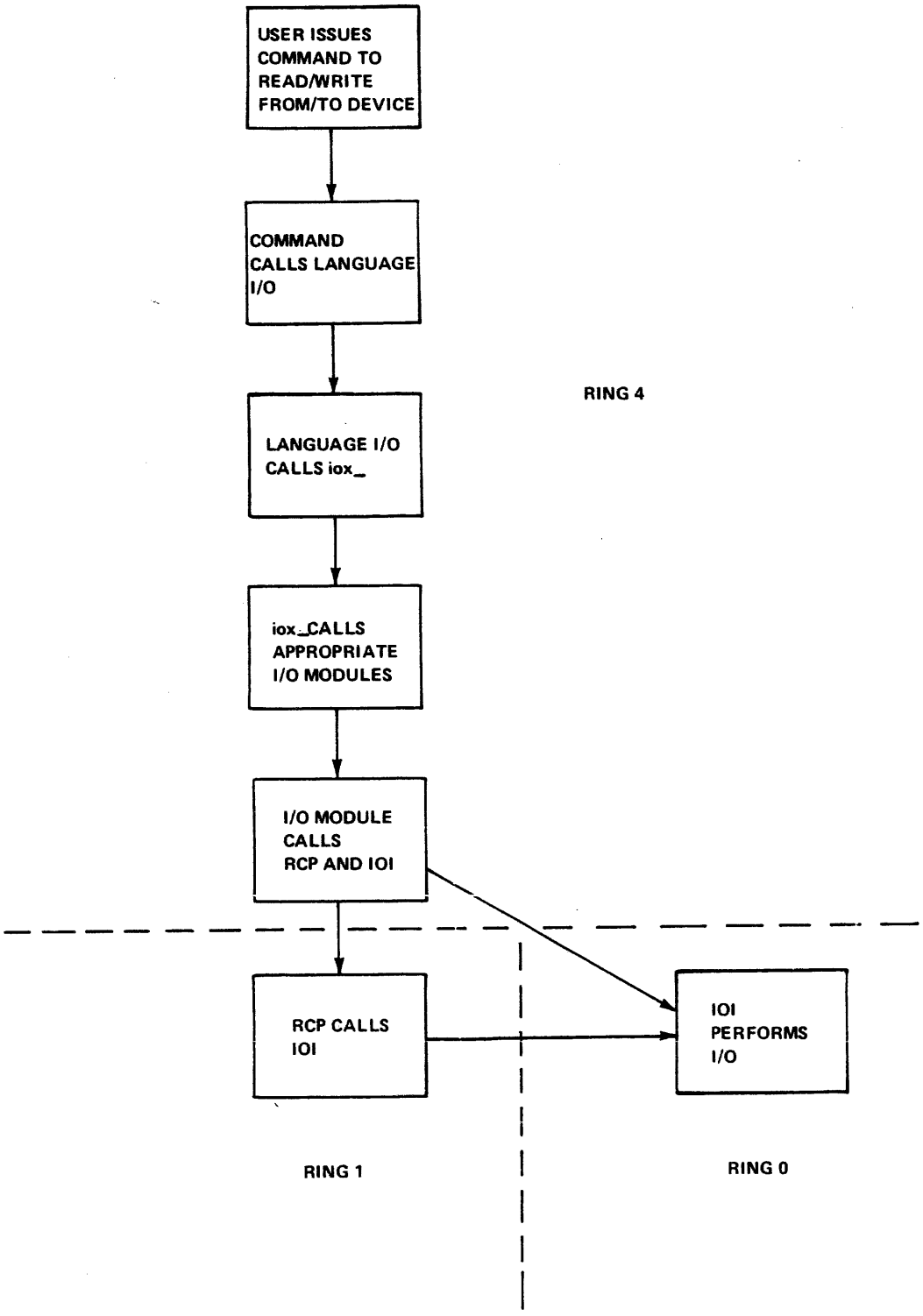


Figure 2-1. Interrelationship between User Code, iox_, RCP,IOI, and the I/O Module.

RESOURCE CONTROL PACKAGE

The function of RCP is to control the access to and usage of I/O devices. RCP executes in ring 1. Access to the various functions of RCP are controlled by the ring 1 gates that must be used to call RCP. One of the primary functions of RCP as a device manager is to control access to IOI. In order to do this, no IOI gate entries are available to perform device attachments, detachments, and other privileged administrative functions. User ring programs, therefore, call RCP in order to request IOI to perform these functions.

An important feature of RCP is its ability to retain registration information for all resources that it controls. It does this by providing administrative interfaces for the registration of resources. Registration of a resource provides information such as: what type of resource this is, what its name is, which attributes it posseses, or in what access class range the resource can be used. Once a resource is registered users may acquire them. The act of acquisition makes a user the owner of the resource--liable for all changes for that resource and in control of discretionary access to the resource (see the acquire_resource command in the MPM Commands).

Another important feature of RCP is its ability to control access to the various resources that it manages (where a resource is either a device or a volume). It does this through the use of access control segments (ACSs). An ACS is a zero length segment whose ACL and ring brackets are used to define the discretionary and intraprocess access to a resource. At a site's discretion, additional features of RCP can be enabled to provide nondiscretionary access control for resources. If this is done, access is also controlled by the AIM access class range of a resource. (See "Access Control" below.)

The resource management functions performed by RCP are:

1. maintain resource information
2. control access to resources
3. reserve and cancel reservation of resources
4. assign and unassign devices
5. attach and detach devices
6. perform special device control functions

The functions reserve, assign and attach are organized into hierarchical levels. Defaults are provided at each level so that users not desiring to exercise features specific to a level do not have to concern themselves with that level.

```
1 reserve
   2 assign
      3 attach
      3 detach
   2 unassign
1 cancel
```

The first level involves the reservation of resources by processes. Reservations are process-specific and remain in effect until the process requests a cancellation or until the process terminates. Reservation implies that a process temporarily has exclusive rights to a resource. This exclusive right means that no other process can use that resource for the duration of the reservation. Reservation does not necessarily imply that a resource is actually being used.

Assignment, like reservation, is process-specific and lasts until unassignment or process termination. An assignment also gives a process temporary exclusive rights to a device. Assignment does not necessarily mean that a device is currently being used. That is the function of the next level, attachment.

A resource cannot be used until it is attached. When RCP is called to attach a resource, it initiates communication with the ring 0 subsystem that actually provides the use of the resource. Before the attachment is completed, RCP performs all initialization necessary to allow the attaching process to begin using the resource. For devices, this involves attaching the device via IOI and making sure that the device is ready and that any volume needed has been mounted.

The hierarchical relationship among reservation, assignment, and attachment implies that the higher-level function, reservation, can stand alone while the lower-level function, attachment, can only be performed after the higher-level function has been performed. RCP can perform the following device reservation, assignment, and attachment functions:

1. Reserving a resource. This means that no other process can use it during this period of time.

2. Explicitly assigning a device. The device is assigned to a process but is not attached.

3. Attaching an explicitly assigned device.

4. Attaching an unassigned device. Since a device cannot be attached until it is assigned; RCP automatically assigns the device and then performs the attachment. The device is said to be implicitly assigned.

5. Detaching an implicitly assigned device. After the device is detached, RCP automatically unassigns the device.

6. Detaching an explicitly assigned device. The device is detached but is not unassigned.

7. Explicitly unassigning a device. If the device is attached, it is first detached and then unassigned.

8. Cancelling reservation of a resource.

The rules stated above imply that I/O modules do not have to be concerned with the assignment or unassignment of devices. They need to be concerned with only the attachment and detachment of a device. RCP, however, does allow the above rules to be overridden. When detaching a device an I/O module can tell RCP to retain the device assignment regardless of whether the device was explicitly or implicitly assigned.

When a process terminates, RCP automatically detaches and unassigns all devices currently assigned to that process and cancels any reservations for that process.

The reservation of resources and cancellation of reservations are done from command level via the reserve_resource and cancel_resource commands and using the -resource control argument with the enter_abs_request command. The explicit assignment and unassignment of devices is done from command level via the assign_resource and unassign_resource commands. The listing of reservations, assignments, and attachments is done from command level via the list_resources command. These commands are described in the MPM Commands.

Device Names

Each device managed by RCP has a unique device name. Device names are derived from the name of the hardware subsystem that controls that type of device. For devices that have exclusive use of a channel, such as printers, the device name is the actual name associated with that channel. For devices that are multiplexed over one or more channels, such as disks, the device name has the form, "ssss_xx", where "ssss" is the subsystem name and "xx" is a device number. Such devices are numbered from 1 to 63. Some examples of device names are:

tapb_03 - tape subsystem B, drive number 3
opc - the operator console
rdrb - a card reader
spc1 - a special new type of device

Access Control

Access to resources is controlled by RCP by first guaranteeing that a user has access to use the resource and then by guaranteeing that the user has not exceeded the per-process limits imposed on certain resources or types of resources.

ACCESS CONTROL SEGMENTS

There are three types of access control on the Multics system: discretionary access control, which is regulated by access control lists (ACL); nondiscretionary access control, which is regulated by the access isolation mechanism (AIM); and intraprocess access control, which is regulated by the ring structure. (For detailed information on types of access, see the MPM Reference Guide.)

An important feature of RCP is its ability to control access to the various resources that it manages. It does this through the use of access control segments (ACSs). An ACS is a zero length segment whose ACL and ring brackets are used to define the discretionary and intraprocess access to a resource. RCP uses an ACS for each resource that it controls; however, an ACS can be shared by more than one resource. The name of an ACS consists of a name, plus the suffix "acs" (e.g., tape_drives.acs). There are no restrictions on ACS names other than the required suffix.

The pathname of the ACS for a resource is specified either at the time the resource is registered or when it is acquired (see the acquire_resource command in the MPM Commands). The specified ACS can later be changed via the set_resource command (see the MPM Commands). If the ACS has not been specified or does not exist, access is set by default to rew for the owner of the resource and null for all other users.

RCP uses the ACS along with other nondiscretionary controls (AIM) to determine the RCP effective access to a resource.

Viewed separately, each type of access control answers the same question, "What access does a particular process have for a particular item?" The access mode granted a process to a resource by discretionary access control (the ACL) is known as the raw access mode.

The way RCP determines effective access to a resource for a process differs from the regular Multics method of determining effective access as follows. First, the effective access to the ACS for the resource is determined as for any segment. If the ACS does not exist, the user appears to have read, execute, and write access if he is the owner of the resource or null access if he is not the owner. Then, two further checks are made. First, the current authorization of the process is compared to the maximum access class of the resource. If write access is not allowed (as defined by the write_allowed_ subroutine) then write and execute access are denied and only read is allowed. Next, the current authorization of the process is compared to the minimum access class of the resource. If read access is not allowed (as defined by the read_allowed_ subroutine) then all access is denied. The resulting access is termed the RCP effective access to the resource. One final restriction enforced by RCP is that, in order to use a device, the RCP effective access must include both read and write.

For example, the following table illustrates some examples of RCP effective access. In the examples below, l1, l2, l3 and l4 represent sensitivity levels and c1, c2, c3, and c4 represent categories.

Table 2-1. RCP Effective Access

Effective Access to ACS	Current Process Authorization	Resource Access Class Range	RCP Effective Access
rew	l1	l1:l3	rew
re	l1	l1:l3	re
rew	l1	l2:l3	null
rew	l3	l2:l3	rew
rw	l4	l2:l3	r
re	l4	l2:l3	r
rw	l2,c1	l1:l4	r
rw	l2,c2	l1,c1:l4,c1,c2	null
rw	l2,c1,c3	l1,c1:l4,c1,c2	r
rw	l2,c1	l1,c1:l4,c1,c2	rw

For more information on AIM, access classes, authorizations, and comparisons involving access classes and authorizations, see the MPM Reference Guide. The pathname of the ACS is specified by the -acs_path control argument and the access class range mentioned above is specified by the -access_class control argument, both of which can be specified in the acquire_resource and set_resource commands (see the MPM Commands).

Manipulating RCP Effective Access

Since the access control mechanisms described above operate together to determine the RCP effective access of a process, there are actions that the user can perform to control this effective access.

First, the user creates an ACS via the create command. Then, the desired ACL for that segment is established using the set_acl command to add desired ACL entries, and the delete_acl command to delete entries. (The above three commands are described in the MPM Commands.) To further affect the ACS, the user may modify its ring brackets by using the set_ring_brackets command (described in the MPM Subsystem Writers' Guide). The system security administrator may set the AIM access class range of the resource itself using the set_resource command (see the MPM Commands).

SITES NOT ENABLING RESOURCE MANAGEMENT

If the system administrator has chosen not to enable Resource Management the preceeding discussion of access control can be simplified.

Nondiscretionary access control is not enforced in this case. There are no ACSs for volumes and all users are assumed to have both read and write access to any volume. The ACS for a device can be found in >system_control_1>rcp and is named device_name.acs (e.g., tape_01.acs). Only the discretionary and intraprocess access (ACL and ring brackets) is considered in determining access to a device.

DEVICE LIMITS

In addition to controlling which processes may have access to a device, RCP will enforce a limit to the number of devices of a given type that a single process may have assigned at one time. This limit is enforced according to the following rules:

1. The limit is not enforced for system processes.

2. The limit for each device type is an installation defined value. They are currently specified on PRPH (peripheral) configuration cards.

3. Currently, only tape drive devices actually have such a limit defined.

RCP will also enforce a limit to the total number of devices of a given type that may be assigned to non-system processes at one time. RCP enforces this limit in order to ensure that a certain number of devices of each device type are either assigned by a system process or available for assignment by a system process. This limit is enforced according to the following rules:

1. The number of devices of each device type that RCP will reserve for system processes are installation defined values. They are currently specified on PRPH configuration cards.

2. Currently, only tape drive devices are reserved for system processes. Only tape drives with certain characteristics are reserved. Only 9 track tape drives are reserved since the backup facility uses only 9 track tapes.

I/O Workspaces

Due to the nature of the Multics virtual memory and its supporting I/O hardware, I/O operations such as "read tape" or "write disk" require all pages of memory referenced by the I/O operation to be in main memory during the operation -- that is, no paging is done during execution of the I/O operation. To accomplish this all channel programs and physical record buffer areas are located in a special segment known as an I/O workspace segment. The ring 0 I/O software, IOI, guarantees that all pages of the workspace are present in main memory before starting the I/O operation and remain there for the duration of the operation.

RCP will control the maximum workspace size associated with each device type. System processes, privileged processes and users on the ACL of the ACS named workspace.acs in the directory >system_control_1>rcp can request up to the privileged maximum workspace size. All others can request up to the normal maximum workspace size. Requests for a workspace longer than is allowed result in an error. The table below lists the workpace maximums that are enforced.

Table 2-2. I/O Workspaces

	Privileged Maximum		Normal Maximum	
device type	words	bytes	words	bytes
tape_drive	45056	180224	3072	12288
disk_drive	45056	180224	2048	8192
printer	45056	180224	1024	4096
punch	45056	180224	1024	4096
reader	45056	180224	1024	4096
special	45056	180224	1024	4096
console	45056	180224	1024	4096

The workspace size is affected by using the -block control argument to those I/O modules that support it. This control argument is used to specify the maximum physical record/block size to be processed. In all cases some overhead for channel programs and I/O module control information must be taken into consideration. When -block is not specified or supported the individual I/O modules choose an appropriate default. In the case of commands that use I/O modules, either the command, some argument or input to the command, or the I/O module may specify/imply in some way the workspace size (for example by supplying -block in an attach description).

Resource Reservation

Users may reserve resources by scheduling with RCP to obtain exclusive rights to a resource for a period of time. RCP enables users to reserve resources or groups of resources through the use of the reserve_resource command (described in the MPM Commands). A reservation takes effect immediately and it lasts until either the user's process is terminated, or the reservation is specifically cancelled via the cancel_resource command (described in the MPM Commands). After invoking reserve_resource, the user has exclusive rights to the resource(s).

Tape volumes, tape drives, disk volumes, and disk drives can be reserved. Tape and disk volumes are specified at the time of reservation by name; tape and disk drives are specified by either name or attributes. In the case of disk drives, the only acceptable attribute is model. For tape drives, acceptable attributes are model, track, and density. Suitable values for the above-mentioned attributes may be found by using the list_resource_types command (also described in the MPM Commands).

To cancel reservations, users invoke the list_resource command to obtain the reservation identifier, and then invoke the cancel_resource command with the reservation identifier to effect the cancellation. Administrators can perform privileged cancellations; that is, if the administrator has proper access, it is possible to cancel reservations belonging to other users.

Device Assignment

The RCP interface for device assignment allows the caller to request the assignment of a specific device, or any appropriate device of a specified type. To request the assignment of a specific device the caller must ask for the device by name. To request the assignment of an appropriate device of a specified type, the caller must specify the characteristics that the assigned device must have. RCP selects a device for assignment based on the following functional algorithm.

1. If the caller has requested a device by name and if this device is already assigned to the calling process, the assignment is aborted.

2. RCP tests all of the devices of the specified type. RCP counts the number of these devices that are appropriate; appropriate and accessible; and appropriate, accessible and available. These requirements are discussed below:

   a. appropriate: A device is considered to be appropriate if it has the device characteristics specified by the caller. In testing each device, RCP does not try to match any device characteristics that are not specified by the caller. If a device is asked for by name, only the device name characteristic is considered.

   b. accessible: A device is considered to be accessible if the calling process has "RW" effective access to the device.

   c. available: A device is considered to be available for assignment if it is not currently assigned to any process or reserved by another process.

3. Having tested each of these requirements, RCP then makes additional tests to see if a device can be assigned. If the assignment cannot be made, RCP returns an error_table_ code that tells the caller why the assignment aborted. The tests that RCP makes at this time are described below:

   a. If there are no appropriate devices, the caller is told that the requested resource (device) is not known to RCP.

   b. If there are no appropriate and accessible devices, the caller is told that he does not have access to the requested resource (device).

   c. If there are no appropriate, accessible and available devices, the caller is told that the requested resource (device) is not available at this time.

   d. If this assignment causes the previously described device limits to be exceeded, the assignment is aborted.

4. If all the tests described above are passed successfully, the device assignment is made. RCP selects the most advantageous device from the list of devices that were found to be appropriate and accessible and available. It makes this selection based on the following rules:

   a. If this is a type of device that has volumes and if the caller specified a volume name to use in the device selection and if any device in the list currently has that volume mounted, RCP selects that device.

   b. If the first case is not true, RCP selects the device that has been idle for the longest amount of time.

Having assigned the device, RCP returns all of the characteristics of this device to the caller.

Device Attachment

Before a device can be attached it must be assigned. The RCP interface for device attachment allows the caller to request a device in the same manner described for device assignment. It can ask for a specific device by name or it can ask for any appropriate device of a specified type. One difference is that if this device is a type that uses volumes, the caller must specify the name of the volume to attach. For assignments, the specification of a volume is optional.

Using the algorithms described above for device assignment, RCP tests all of the devices of the specified type that are already assigned by the requesting process. If the specific device or any appropriate device is already assigned to this process, RCP attaches that device. If no suitable device is already assigned to the requesting process, RCP automatically attempts to assign a suitable device to this process. If no device can be assigned then the attachment is aborted. If the attachment is for a device type that uses volumes, RCP checks to see if the specified volume is already attached to this process or any other process. If the volume is already attached, RCP aborts the attachment.

Once RCP has found a suitable assigned device, it begins the real work of attaching the device. This involves calling IOI to perform the ring 0 device attachment. If the device is a type that uses volumes, RCP tells the operator to mount the specified volume if it is not already mounted on the proper device. Before the attachment is completed, RCP makes sure that the volume has been mounted and that the write protection mechanism provided by the device is set correctly. When all of this initialization work has been completed, RCP calls IOI to set the workspace and time-out limits and to promote the validation level of the device. Until this is done, the IOI validation level for the device is the RCP validation level (1). Thus no program in a higher ring can successfully call IOI to use this device until RCP tells IOI to promote it. RCP returns all of the device characteristics of the attached device and all of the information needed to communicate with IOI about this device.

I/O INTERFACER

The I/O interfacer (IOI) allows user-ring programs to perform peripheral I/O. It is used by all user-ring programs that perform I/O to devices connected to the Input/Output Multiplexer (IOM) channels. The user can construct device-specific DCW lists and call IOI to initiate the I/O operation. When the operation completes, IOI provides the user with a wakeup and the status. The hardware protection and relocation features of the IOM are used by IOI to allow the user complete control over his DCW lists and data with no possibility of damaging the system.

NOTE: More information on the IOI will be supplied in a future update of this manual.

SECTION 3

MULTICS STANDARD TAPE FORMAT

This section describes the standard physical format used on 7-track and 9-track magnetic tapes on the Multics system. This format is known as Multics standard tape format. Tapes of this form may be written and read by the tape_mult_ I/O module (described in Section 5). Any magnetic tape not written in the standard format described here is not a Multics standard tape.

STANDARD TAPE FORMAT

The first record on the tape following the beginning of tape (BOT) mark is the tape label record. Following the tape label record is an end of file (EOF) mark. Subsequent reels of a multireel sequence also have a tape label followed by EOF. (An EOF mark is the standard sequence of bits on a tape that is recognized as an EOF by the hardware.)

Following the tape label and its associated EOF are the data records. An EOF is written after every 128 data records with the objective of increasing the reliability and efficiency of reading and positioning within a logical tape. Records that are repeated because of transmission, parity, or other data alerts, are not included in the count of 128 records. The first record following the EOF has a physical record count of 0 mod 128.

An end of reel (EOR) sequence is written at the end of recorded data. An EOR sequence is:

EOF mark
EOR record
EOF mark
EOF mark

STANDARD RECORD FORMAT

Each physical record (with the exception of the tape label record) consists of a 1024-word (36864-bit) data space enclosed by an 8-word header and an 8-word trailer. The total record length is then 1040 words (37440 bits). The header and trailer are each 288 bits. This physical record requires 4680 frames on 9-track tape and 6240 frames on 7-track tape. This is approximately 5.85 inches on 9-track tape at 800 bpi and 7.8 inches on 7-track tape at 800 bpi, not including interrecord gaps. (Record gaps on 9-track tapes are approximately 0.6 inches and on 7-track tapes are approximately 0.75 inches, at 800 bpi.)

For 1600 bpi 9-track tape, the record length is approximately 2.925 inches (with an interrecord gap of approximately 0.5 inches).

PHYSICAL RECORD HEADER

The following is the format of the physical record header:

Word 0: Constant with octal representation 670314355245.

Words 1 and 2: Multics standard unique identifier (70 bits, left justified). Each record has a different unique identifier.

Word 3: Bits 0-17: the number of this physical record in this physical file, beginning with record 0.

Bits 18-35: the number of this physical file on this physical reel, beginning with file 0.

Word 4: Bits 0-17: the number of data bits in the data space, not including padding.

Bits 18-35: the total number of bits in the data space. (This should be a constant equal to 36864.)

Word 5: Flags indicating the type of record. Bits are assigned considering the leftmost bit to be bit 0 and the rightmost bit to be bit 35. Word 5 also contains a count of the rewrite attempt, if any.

Bit	Meaning if Bit is 1
0	This is an administrative record (one of bits 1 through 13 is 1).
1	This is a label record.
2	This is an end of reel (EOR) record.
3-13	Reserved.
14	One or more of bits 15-26 are set.
15	This record is a rewritten record.
16	This record contains padding.
17	This record was written following a hardware end of tape (EOT) condition.
18	This record was written synchronously; i.e., control did not return to the caller until the record was written out.
19	The logical tape continues on another reel (defined only for an EOR record).
20-26	Reserved.
27-35	If bits 14 and 15 are 1, this quantity indicates the number of the attempt to rewrite this record. If bit 15 is 0, this quantity must be 0.

Word 6: Contains the checksum of the header and trailer excluding word 6; i.e., excluding the checksum word.

Word 7: Constant with octal representation 512556146073.

PHYSICAL RECORD TRAILER

The following is the format of the trailer:

Word 0:	Constant with octal representation 107463422532.
Words 1 and 2:	Standard Multics unique identifier (duplicate of header).
Word 3:	Total cumulative number of data bits for this logical tape (not including padding and administrative records).
Word 4:	Padding bit pattern (described below).
Word 5:	Bits 0-11: reel sequence number (multireel number), beginning with reel 0. Bits 12-35: physical file number, beginning with physical file 0 of reel 0.
Word 6:	The number of the physical record for this logical tape, beginning with record 0.
Word 7:	Constant with octal representation 265221631704.

NOTE: The octal constants listed above were chosen to form elements of a single-error-correcting code whether read as 8-bit tape characters (9-track tape) or as 6-bit tape characters (7-track tape).

ADMINISTRATIVE RECORDS

The standard tape format includes three types of administrative records: a standard tape label record, a bootable tape label record, and an end of reel (EOR) record.

Standard Tape Label Record

The standard tape label record is written in standard record format, and can best be defined by the PL1 structure declaration that follows:

```
dcl 1 stand_label_record      based (mstrp) aligned,
    2 head                    like mstr_header,
    2 installation_id         char (32),
    2 tape_reel_id            char (32),
    2 volume_set_id           char (32),
    2 pad (1000)              bit (36),
    2 trail                   like mstr_trailer;
```

where:

1. head
   is the standard 8-word record header described above.

2. installation_id
   is the ASCII installation code. This identifies the installation that labeled the tape.

3. tape_reel_id
   is the ASCII reel identification. This is the reel identification by which the operator stores and retrieves the tape.

4. volume_set_id
is the name of the volume set if the "-volume_set_name" tape_mult_ attach description argument was used when the tape reel was created. If the "-volume_set_name" attach description argument was not used, this field is padded with ASCII blanks.

5. pad
is an array of words containing the standard padding pattern (described below), used to fill the label record data space to the standard size.

6. trail
is the standard 8-word record trailer described above.

Bootable Tape Label Record

The bootable tape label record is an administrative record, written in nonstandard format. The first eight words of the physical record contain four pairs of executable instructions collectively known as a transfer vector. This transfer vector allows a Multics standard tape to be bootloaded from any of four possible I/O controllers.

When a tape that contains a bootable tape label record is bootloaded, a hardwired program within the I/O controller writes the data within the first record starting at location 30 (octal, absolute) in memory. When the data transfer is completed, the I/O controller sets an interrupt "cell" in the system controller, which causes the bootload processor to execute a hardwired "XED" instruction to the address indicated by the system controller. This interrupt address generated by the system controller is a function of the interrupt "cell" set by the I/O controller and by the configuration panel number of the I/O controller itself. For example, if the bootload sequence was initiated on I/O controller #0, then the interrupt address would be 30 (8); addresses 32, 34, and 36, respectively, would be generated by I/O controllers number 1, 2, and 3. The executable instructions contained in each pair of the transfer vector are:

```
lda     4
tra     330
```

Location 4 contains the DCW address stored by the I/O controller hardwired boot program. An executable program is located at 330 (octal, absolute). This program is known as the tape label boot program.

The bootable tape label record is created through the use of the tape_mult_ boot_program control order. This control order is normally executed by the generate_mst command to write a bootable label on BOS system tapes. Although a user can write his own boot program and have generate_mst write it to the BOS tape label, a standard boot label exists in the system libraries, named mst_boot_label.

The mst_boot_label boot program initializes the bootstrap environment and sets up an I/O channel program to read and skip the EOF record, and to read in the first data record on the tape under control of a DCW. The DCW address used is 7750 (8) absolute with a word count of 4096. (The generate_mst command places the standard 8-word tape record header plus a 16-word segment header before the first data in the record; the first executable data in the record starts at location 10000 (8).) After the first data record is read in, the status returned from the tape controller is checked for errors. If an error occurred, the status word is copied in the A register and the processor falls into a DIS. Assuming no status error was detected, control is transferred to absolute location 10000 (8).

There are many other fields in the bootable tape label record. The following is a PL1 structure declaration of the contents of the bootable tape label record followed by an explanation of each field:

```
dcl 1 mst_label             based (mstrp) aligned,
    2 xfer_vector           (4),
      3 lda_instr           bit (36),
      3 tra_instr           bit (36),
    2 head                  like mstr_header,
    2 installation_id       char (32) unaligned,
    2 tape_reel_id          char (32) unaligned,
    2 volume_set_id         char (32),
    2 fv_overlay            (32) unaligned,
      3 scu_instr           bit (36),
      3 dis_instr           bit (36),
    2 fault_data (8)        bit (36),
    2 boot_pgm_path         char (168) unaligned,
    2 userid_char           (32) unaligned,
    2 label_version         fixed bin,
    2 output_mode           fixed bin,
    2 boot_pgm_len          fixed bin,
    2 copyright             char (56) unaligned,
    2 pad (13)              bit (36),
    2 boot_pgm              (0 refer (mst_label.boot_pgm_len)) bit (36),
    2 trail                 like mstr_trailer;
```

where:

1. xfer_vector
   is the bootload transfer vector. There is one transfer vector for each of four possible I/O controllers. The transfer vector functions to gain control as the result of an interrupt after a bootload sequence.

2. lda_instr
   is an "LDA" instruction from absolute location 4, which for an IOM is the payload channel DCW as stored by the hardwired bootload program in the IOM.

3. tra_instr
   is an unconditional transfer to the beginning of the bootload program.

4. head
   is the standard 8-word record header described above.

5. installation_id
   is the ASCII installation code. This identifies the installation that labeled the tape.

6. tape_reel_id
   is the ASCII reel identification. This is the reel identification by which the operator stores and retrieves the tape.

7. volume_set_id
   is the name of the volume set if the "-volume_set_name" tape_mult_ attach description argument was used when the tape reel was created. If the "-volume_set_name" attach description argument was not used, this field is padded with ASCII blanks.

8. fv_overlay
   This 32-element array overlays the hardware fault vector area at absolute location 100 (octal) if this tape is bootloaded. If an unexpected fault occurs when this tape is bootloaded, the appropriate fault pair is executed by the processor fault logic.

9. scu_instr is a Store Control Unit (SCU) instruction, which safe-stores the state of the processor control unit when executed.

10. dis_instr
is an interrupt inhibited Delay until Interrupt Signal (DIS) instruction, which halts the processor when executed.

11. fault_data
is an area where SCU data is stored if an unexpected fault ocurred while bootloading this tape.

12. boot_pgm_path
if nonblank can be the absolute pathname of the boot program written on this label record. It can also be the user designated name for the boot program when the "boot_program" tape_mult_ control order was executed.

13. userid
is the User_id (Person.Project.Instance) of the user who created this tape.

14. label_version
is the version number of this label record structure, currently 2.

15. output_mode
is the number of the iox_ mode in effect when this tape was created. (See iox_modes.incl.pl1.)

16. boot_pgm_len
is the length of the boot program in words. The boot program must be less than or equal to 840 (1510 octal) words in length. If it is less than 840 words, the record is padded out with the standard padding pattern.

17. boot_pgm
is the executable text of the boot program. The boot program must be coded in absolute self-relocating ALM assembly language.

18. trail
is the standard 8-word record trailer described above.

End of Reel Record

The end of reel record contains only padding bits in its data space. The standard record header of the EOR record contains the information that identifies it as an EOR record (word 5, bits 0 and 2 are 1).

DENSITY AND PARITY

Both 9-track and 7-track standard tapes are recorded in binary mode with odd ones having lateral parity. Standard densities are 800 frames per inch (bpi) (recorded in NRZI mode), 1600 bpi (recorded in PE mode), and 6250 bpi (recorded in GCR mode).

DATA PADDING

The padding bit pattern is used to fill administrative records and the last data record of a reel sequence.

WRITE ERROR RECOVERY

Multics standard tape error recovery procedures differ from conventional techniques in that no attempt is made to backspace the tape on write errors. If a data alert occurs while writing a record, the record is rewritten. If an error occurs while re-writing the record, that record is again rewritten. Up to 64 attempts can be made to write the record. No backspace record operation is performed.

The above write error recovery procedure is applied to both administrative records and data records.

COMPATIBILITY CONSIDERATION

The software is capable of reading Multics standard tapes that are written with records with less than 1024 words in their data space. In particular, a previous Multics standard tape format specified a 256-word (9216-bit) data space in a tape record.

In addition to recognizing and reading standard and bootable tape label records, the software is also capable of recognizing and reading Multics standard tapes that were generated with a version 1 label record, i.e., standard label records that do not contain the volume_set_id field.

SECTION 4

COMMANDS

This section contains descriptions of tape-related Multics commands, presented in alphabetic order. Each description contains the name of the command (including the abbreviated form, if any), discusses the purpose of the command, and shows the correct usage. Notes and examples are included when deemed necessary for clarity.

The commands described in this section and their functions are:

copy_file	copies records from an input file to an output file
list_tape_contents	prints information about files recorded on 9-track magnetic tape
read_tape_and_query	allows the user to interactively inspect the contents of a magnetic tape
tape_in	transfers files between magnetic tape and the storage system
tape_out	transfers files between the storage system and magnetic tape

Also refer to the assign_resource, list_resources, and unassign_resource commands in the MPM Commands. These commands deal with the resource control package and the consignment of devices.

Name: copy_file, cpf

The copy_file command copies records from an input file to an output file (both files reside in memory). The input and output file records must be structured. (See "Unstructured Files" below for an explanation of how unstructured files can be copied.) The input file can be copied either partially or in its entirety.

The copy command makes an exact duplicate of the input file, whereas copy_file produces an output file that has been restructured for maximum compactness. (See the description of the copy command in the MPM Commands.)

Usage

copy_file in_control_arg out_control_arg {-control_args}

where:

1. in_control_arg
   is one of two input control arguments that specifies the input file from which records are read. The file may be specified by either an I/O switch name or an attach description. (See "Notes" below.)

   -input_switch STR, -isw STR
   specifies the input file by means of an already attached I/O switch name, where STR is the switch name.

   -input_description STR, -ids STR
   specifies the input file by means of an attach description, where STR is the attach description. The attach description string must be enclosed in quotes if it contains spaces.

2. out_control_arg
   is one of two output control arguments that specifies the output file to which these records are written. It may be either an I/O switch name or an attach description. (See "Notes" below.)

   -output_switch STR, -osw STR
   specifies the output file by means of an already attached I/O switch name, where STR is the switch name.

   -output_description STR, -ods STR
   specifies the output file by means of an attach description, where STR is the attach description. The attach description string must be enclosed in quotes if it contains spaces.

3. control_args
   may be one or more of the following control arguments. (See "Notes" below.)

   -keyed
   copies both records and keys from a keyed sequential input file to a keyed sequential output file. The default is to copy records from an input file (either keyed or not) to a sequential output file. (See "Keyed Files" below.)

-from N, -fm N
copies records beginning with the Nth record of the input file, where N is a positive integer. The default is to begin copying with the "next record." (See "Notes" below.)

-start STR, -sr STR
copies records beginning with the record whose key is STR, where STR is 256 or fewer ASCII characters. The default is to begin copying with the "next record."

-to N
copies until the Nth record has been copied or the input file is exhausted, whichever occurs first, where N is a positive integer greater than or equal to the N given with the -from control argument. This control argument can only be specified if -from is also specified. The default is to perform copying until the input file is exhausted.

-stop STR, -sp STR
copies until the record whose key is STR has been copied or the input file is exhausted, whichever occurs first, where STR is 256 or fewer ASCII characters. This control argument can be specified without specifying the -start control argument. However, if -start is specified, the STR given with -stop must be greater than or equal to (according to the ASCII collating sequence) the STR given with -start. The default is to perform copying until the input file is exhausted.

-count N, -ct N
copies until N records have been copied or the input file is exhausted, whichever occurs first, where N is a positive integer. The default is to perform copying until the input file is exhausted.

-all, -a
copies until the input file is exhausted. This is the default.

-brief, -bf
suppresses an informative message indicating the number of records actually copied.

-long, -lg
prints an informative message indicating the number of records actually copied. This is the default.

Unstructured Files

The copy_file command operates by performing record I/O on structured files. If it is desired to copy from/to an unstructured file, the record_stream_ I/O module can be used, e.g., by typing the command line:

```
cpf -ids "record_stream_ -target vfile_ pathname" -osw OUT
```

The effect is to take lines from the file specified by pathname via the vfile_ I/O module, transform them into records via the record_stream_ I/O module, and then copy them to the I/O switch named OUT.

Keyed Files

The copy_file command can copy a keyed sequential file either as such, or as though it were purely sequential. By default, the command copies only records and does not place keys in the output file. To copy the keys, the -keyed control argument must be used. When -keyed is used, the input file must be a keyed sequential file. Whether keys are copied or not, control arguments can be used to delimit the range of records to be copied (i.e., -start, -stop, -from, -to, -count). Copying is always performed in key order.

Notes

If either the input or output specification is an attach description, it is used to attach a uniquely named I/O switch to the file. The switch is opened, the copy performed, and then the switch is closed and detached. Alternately, the input or output file may be specified by an I/O switch name. Either the io_call command or iox_ subroutine may be used to attach the file prior to the invocation of the copy_file command. (See the description of the io_call command in the MPM Commands and the iox_ subroutine in the MPM Subroutines.)

If the input file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_input, performs the copy, and closes it. If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_input, sequential_input_output, keyed_sequential_input, or keyed_sequential_update. The switch is not closed after the copy has been performed.

The "next record" must be defined if neither the -start nor -from control argument is used to specify an absolute starting position within the input file. If the I/O switch is opened by the copy_file command, the next record is the first record of the file; otherwise, the next record is that record at which the file is positioned when the copy_file command is invoked.

If the output file is specified by an I/O switch name and the switch is not open, the copy_file command opens it for (keyed_)sequential_output, performs the copy, and closes it. If the switch is already open when the copy_file command is invoked, the opening mode must be sequential_output, sequential_input_output, keyed_sequential_output, keyed_sequential_update, direct_output, or direct_update. (In update mode, output file records with keys that duplicate input file records are rewritten.) The switch is not closed after the copy has been performed.

The -from and -start control arguments are mutually exclusive. The -to, -stop, -count, and -all control arguments are mutually exclusive. The -brief and -long control arguments are mutually exclusive. The informative message, printed by default, appears as one of the following:

345 records copied.

Examples

To copy an entire file from an already attached file to the segment in_copy, type:

```
cpf -isw in -ods "vfile_ in_copy"
```

To copy the first 13 records from a tape file to an output file, the two lines below would actually be typed as only one per line (The normal result of this command would be to print the first 13 records on the user's terminal.)

```
cpf -ct 13 -ids "tape_ansi_ 887677 -name TEST21 -ret all"
  -ods "record_stream_ user_output"
```

To copy 13 records from an already attached file to another already attached file, starting with the 56th record of the input file, type:

```
cpf -isw in -osw out -from 56 -ct 13
```

To copy records 43 through 78 from an already attached file to an already attached file, type:

```
cpf -isw in -osw out -from 43 -to 78
```

To copy all but the first seven records from segment testdata.11 to an already attached file, type:

```
cpf -ids "vfile_ testdata.11" -osw out -fm 8
```

To copy an entire keyed sequential file with keys, type:

```
cpf -isw in -osw out -all -keyed
```

To copy 13 records of a keyed sequential file starting with the record whose key is ASD66 to a sequential output file, the following line is typed. (No keys are copied.)

```
cpf -isw in -osw out -sr ASD66 -ct 13
```

To copy the records and keys from a keyed sequential file up to and including the record whose key is bb"bb, type:

```
cpf -keyed -isw in -osw out -sp "bb""bb"
```

list_tape_contents

Name: list_tape_contents, ltc

The list_tape_contents command prints information about files recorded on 9-track magnetic tape. Tapes that may be listed include ANSI standard labeled tapes and IBM standard labeled tapes (see the tape_ansi_ and tape_ibm_ I/O modules in Section 5).

The information printed by this command is extracted from the tape labels and printed in various amounts according to the control arguments supplied. Where information is not obtainable from the label, the value "****" is printed as the item entry. Three printing modes are available to the user: long mode, which prints extensive information about the files on a tape; brief mode, which prints only the basic information about the files on a tape; and default mode, which prints slightly more information than does brief mode.

Usage

list_tape_contents {volume_name {-comment STR}} {-control_args}

where:

1. volume_name
   is the volume name specification of the tape volume or volume-set to be listed. A maximum of 64 volumes may be specified in this list. The keyword -volume or -vol must precede the volume_name if the volume_name begins with a hyphen (-); otherwise, -volume is optional. For tapes written on the Multics system, only the first volume name of the volume-set need be given, since the I/O module determines the other members of the set from file labels. However, for tapes written on other systems, all of the volume names of the volume-set must be given. See "Volume Specification" in the tape_ansi_ and tape_ibm_ I/O module descriptions.

2. -comment STR, -com STR
   displays a message on the operator's console when the volume volume_name immediately preceding the the -comment keyword is mounted. STR is a string of from 1 to 64 characters comprising the message to be displayed.

3. control_args
   can be chosen from the following and can appear only once in the command line:

   -long, -lg
   prints an extensive amount of information about files on an OS or ANSI standard labeled tape. This file information includes: the file identifier (Id:), the file sequence number (Number:), the record format (Format:), the physical block size, in characters (Blksize:), the logical record length, in characters (Lrecl:), the encoding mode (Mode:), the file creation date (Created:), the file expiration date (Expires:), the file-set section number (Section:), the file version number (Version:), the file generation number (Generation:), and the operating system that recorded the tape (System:). (See "Examples" below.)

list_tape_contents

-brief, -bf
prints a brief amount of information about each file on an ANSI standard or OS (see tape_ibm_ in Section 5) standard labeled tape. The file information listed in brief printing mode is just the file identifier (Id:) and the file sequence number (Number:). (See "Examples" below.)

-io_module STR, -iom STR
invokes a system I/O module to attach and read the specified tape volume. Only the tape_ansi_ or tape_ibm_ I/O modules are valid specifications here. The tape_ansi_ subroutine is specified in order to list ANSI Standard labeled tape, the tape_ibm_ subroutine is specified in order to list OS standard labeled tapes. The tape_ansi_ subroutine is the default I/O module if this argument is omitted.

-to N
specifies that only the first N files on a tape are to be listed, where N is an integer such that $1 \leq N \leq 9999$. If fewer than N files exist on the tape, a warning message to this effect is printed.

-retain STR, -ret STR
specifies which resources are to be retained on termination of this command. STR can be the string all or the string none. If STR is not specified, the string none is assumed.

Error Processing

If an unrecoverable error occurs during volume processing, further I/O is not possible and the listing of the tape is terminated.

Notes

The -long and -brief control arguments are mutually exclusive.

To obtain information about the volume-set, list_tape_contents creates an attach description which is used to attach each file. This attach description is the catenation of the following in the order specified: the I/O module name; volume name, -comment STR pairs and any unrecognized argument; and finally a string of the form "-retain all -number n" where n is the file number to be processed. Note that due to this last string list_tape_contents will not allow -number or -retain arguments in the command line.

If neither the -long nor -brief control arguments are specified in the command line for listing an OS or ANSI standard labeled tape, the default action is to list somewhat more file information than that printed for the -brief control argument. This file information includes: the file identifier (Id:), the file sequence number (Number:), the record format (Format:), the block size, in characters (Blksize:), the logical record length, in characters (Lrecl:), the file creation date (Created:), and the file expiration date (Expires:).

list_tape_contents

Examples

```
! list_tape_contents 070065 -iom tape_ansi_ -to 3

    Mounting volume 070065 with no write ring.
    070065 mounted on tape_05.

    File listing of ANSI Labeled Volume 070065 Recorded at 1600 bpi.

    Id:     Number:   Format:   Blksize:   Lrecl:   Mode:   Created:   Expires:
    FILE1      1         DB        8192      137    ansi    09/02/76   unknown
    FILE2      2
    FILE3      3

    Displayed characteristics for the last 3 files are identical
```

The above example lists only the first three files on tape volume 070065. The file information is displayed in default printing mode since no verbosity control arguments are given in the command line. After the mount message, a header is printed identifing the tape as an ANSI standard labeled tape recorded at a density of 1600 bpi. Since the recording characteristics for all of the files on the tape are the same, these are only printed once for the first file.

```
! list_tape_contents 070065 -iom tape_ansi_ -brief

    Mounting volume 070065 with no write ring.
    070065 mounted on tape_02.

    File listing of ANSI Labeled Volume 070065 Recorded at 1600 bpi.

    Id:       Number:
    FILE1        1
    FILE2        2
    FILE3        3
    FILE4        4
    FILE5        5
    FILE6        6
    FILE7        7
```

This example lists the entire contents of tape volume 070065 in brief printing mode. Again, after the mount message, a header is printed giving the tape type as ANSI standard labeled and the recording density as 1600 bpi. Brief printing mode gives only the file identifier and the file number for each file listed.

read_tape_and_query

Name: read_tape_and_query, rtq

The read_tape_and_query command allows a user to interactively inspect and determine the contents of a magnetic tape. Physical tape file processing capabilities are also provided.

Usage

read_tape_and_query volume_id {-control_args}

where:

1. volume_id
   is the local tape library designation of the requested tape volume.

2. control_arg
   can be chosen from the following:

   -block N, -bk N
   specifies the maximum physical record size to be processed, where N is the number of bytes. The default is 11200 bytes (2800 36-bit words).

   -comment STR, -com STR
   displays STR as a message on the operators console at the time that tape volume <volume_id> is mounted. If STR contains spaces, tabs or special characters, the entire STR must be enclosed in quotes.

   -density N, -den N
   specifies the initial density setting for tape attachment, where N is the number of bits per inch (bpi). The default is 800 bpi. Although the density is automatically determined (see "Notes" below), some tape subsystems may not have tape drives capable of handling the default density.

   -no_prompt
   suppresses printing of the prompt character string ("-->") when at rtq command level.

   -ring, -rg
   specifies that the tape is to be mounted with a write ring. This allows a tape that is already mounted with a write ring to be attached without operator intervention. The default is to mount the tape with no write ring.

   -track N, -tk N
   where N is 7 or 9 for 7 or 9 track tapes. If this control argument is not specified, 9 track is assumed.

read_tape_and_query

Notes

The read_tape_and_query command requests the specified tape volume to be mounted. After the mount request has been satisfied, read_tape_and_query automatically determines the tape density and checks for a recorded tape label. If the density can be determined, an informative message is displayed that includes the density. If the tape has a standard Multics, GCOS, IBM, ANSI or CP5 tape label, an informative message is displayed that includes the standard label type and the recorded volume name. If the tape contains a valid IBM or ANSI label, a second message is displayed informing the user of the physical block size and logical record length (in bytes) of the first data file. For all standard labeled tape volumes, the tape is then positioned to the beginning of the first data file. If the tape label is not recognized as one of the five standard types mentioned above, it is designated as unlabeled and the tape volume is repositioned to the beginning of the tape. The read_tape_and_query command then goes into a request loop after displaying a prompt character string ("-->"), unless -no_prompt has been specified. Some requests acceptable to read_tape_and_query take arguments that are optional. These optional arguments are enclosed in braces. The valid user responses while in this request loop are as follows:

quit, q
: detaches the tape and returns control to the current command processor.

help, ?
: lists the requests of read_tape_and_query.

.. <rest_of_line>
: passes <rest_of_line> to the command processor for execution as a Multics command.

.
: displays the command name read_tape_and_query with its short name (rtq) in parentheses.

position, pos
: displays the current physical tape file and record position for the user.

bsr {N}
: backspace N records. If N is not specified, 1 is assumed.

bsf {N}
: backspace N files. If N is not specified, 1 is assumed.

bof
: position to the beginning of the current physical tape file.

fsr {N}
: forward space N records. If N is not specified, 1 is assumed.

fsf {N}
: forward space N files. If N is not specified, 1 is assumed.

read_tape_and_query

rewind, rew
: issues a rewind command and positions the tape to the beginning of tape (BOT) marker.

density <N>, den <N>
: sets the tape density to <N> bits per inch (bpi), where N can be 6250, 1600, 800, 556 or 200. Density requests must be issued while the tape is positioned at the BOT marker or a request reject status results. It is not normally necessary to set the tape density as it is automatically set by read_tape_and_query before the request loop is entered.

mode STR
: sets the hardware mode for reading tape to STR, which can be one of the following modes:

  bin
  : eight bit bytes are read in and packed (nine eight bit bytes per memory double word). This is the default mode.

  bcd
  : reads in tape that was originally written in binary coded decimal (BCD). The hardware performs input character conversion.

  nine
  : eight bit bytes are read in and converted to nine bit bytes by forcing the most significant bit of each nine bit byte to "0"b.

read_record {-count N}, rdrec {-ct N}
: reads the current record into a temporary buffer. If the tape is one of the five known standard labeled tapes, the record is checked to determine if it is a label or trailer record; if it is, information pertinent to that particular record type is displayed. Otherwise, information pertaining to the physical record length in bits, words, 8-bit bytes, 9-bit bytes, and 6-bit characters is displayed. When the -count argument is specified, N records are read, overlaying each other in the temporary buffer.

list_tape_contents {-long} {-label}, ltc {-lg} {-lbl}
: displays information about each record on the tape. The tape is positioned to BOT and each record is read in. If the tape is one of the five known standard types, the current record is inspected to determine if it is a valid label or trailer record; if so, information pertinent to that particular label or trailer record is displayed, in interpreted format. If the -long argument is used, the contents of the label record is displayed (in ASCII) as well. Otherwise, the length of the current record is compared to the length of the last record read. If the lengths are the same, a tally of the number of records with the same length is incremented. If the length of the current record is different from that of the last record, or if an end of file mark is detected, a message is displayed that includes: the number of records of equal length, and the record length in bits, words, 8-bit bytes, 9-bit bytes, and 6-bit characters. This display of record lengths can be circumvented by using the -label argument, which only displays the label records. This operation continues until the logical end of tape is reached (two end of file marks in succession or an end of volume trailer record, followed by an end of file mark). The tape is repositioned to BOT after the list_tape_contents request is complete. Use of the -label argument with unlabeled tapes is treated as an error.

read_tape_and_query

dump {offset} {n_words} {char_types}
displays the contents of the record buffer (filled with the read_record request) on the users terminal. If no arguments are specified, the contents of the entire tape buffer are displayed in octal format. If the n_words argument is specified, it must follow offset. However, these arguments may be positioned before or after any char_type arguments that may be specified. The offset and n_words arguments must be specified in octal. If offset is specified without being followed by n_words, then the tape buffer is dumped starting with the <offset>th word and ending with the last word in the tape buffer. The char_type optional arguments allow interpretation of the data contained in the tape buffer in various character formats. If more than one char_type argument is specified, then the tape buffer is dumped with the first character interpretation, followed by the next character interpretation, and so on until all requested data formats have been dumped. The value of char_type can be selected from the following:

-ascii
displays the contents of the record buffer in octal with an ASCII interpretation of the data on the right side.

-bcd
displays the contents of the record buffer in octal with a BCD interpretation of the data on the right side

-ebcdic
displays the contents of the record buffer in octal with an EBCDIC interpretation of the data on the right side.

-hex
displays the record buffer in hexadecimal format.

read_file {args}, rdfile {args}
reads the current tape file into the segment described by args. The default action of this request with no arguments queries the user as to the segment name he wishes the tape file to be read into and then issues a warning telling the user that the current tape file will be read in as a stream file with no conversion. The user is asked if he wishes to continue. If he answers yes, then the tape file is read into the designated segment and a newline character is appended to each physical record. If the user answers no, then control is returned to the request loop. If the tape is one of the five standard types, each record is checked to determine if it is a valid label or trailer record. If it is, pertinent information about the record is displayed and the record is not written to the output segment.

The optional arguments associated with the read_file request are:

-output_file {STR}, -of {STR}
where STR specifies the segment name for the tape file to be read into. If STR is omitted, the user is queried for the segment name.

read_tape_and_query

-count N, -ct N
allows reading up to N files, or until logical end of tape is encountered. After the first file is read in, the -count iteration count is appended to the end of the user-designated output file name as a second component. For example:

```
rdfile -ct 3 -of file1
```

names the first output file file1, the second file1.2, and the third file1.3.

-multics, -mult
specifies that the input tape file is in Multics standard system format. (Refer to Section 3 in this manual for a description of Multics standard tape format.) The data portion of each unrepeated record is written to the specified stream output file. No attempt is made to separate the contents of the physical record into a logical format. Since standard Multics tape format specifies that an EOF mark be written every 128 records, the "-extend" and "-count" arguments should be used to ensure that all of the data is recovered.

-gcos, -gc
specifies that the input tape file is in GCOS standard system format. That is, each record has a block control word and several record control words dividing the physical record into logical records. Each record is processed accordingly. BCD records are converted to ASCII. ASCII records are copied directly. Binary compressed deck card images are decompressed and converted to ASCII. If a BCD card image is identified as a "$ object" card, this card image and all successive binary card images, until a "$ dkend" card image is identified, are copied to a separate file whose name is formed from columns 73 - 76 of the $ object card with a suffix of ".obj". If a BCD card image is identified as a "$ snumb" card, this card and all following card images, until another $ snumb card or end of file, are copied into a file whose name is formed from columns 16 - 21 of the $ snumb card with a suffix of ".imcv". If a BCD card image is identified as a "$ <language>" card, this card and all following card images, until another $ <language> card or end of file, are copied into a file whose name is formed from columns 73 - 76 of the $ <language> card with a suffix of ".ascii". This file is also surrounded by sufficient GCOS "JCL cards" so that the completed "deck" can be assembled using the Multics GCOS Environment Simulator. If columns 73 - 76 of the $ <language> card are blank, the $ <language> card image is displayed and the user is queried for the filename.

-cp5
specifies that the input tape file is in CP5 standard system format, which consists of variable length records, recorded in EBCDIC. Each variable length logical record is written to the specified stream file, with a newline character appended to the end. The data read from the tape is automatically converted from EBCDIC to ASCII.

read_tape_and_query

-dec
specifies that the input tape file is in Digital Equipment Corporation (DEC) standard system format. Each DEC word is 40 bits long, of which the first 32 bits and the last 4 bits are concatenated together to form one 36 bit word. The other 4 bits are discarded. The converted data is then written to the specified file in raw format.

-ibm_vb {STR}
specifies that the input tape file is standard IBM "VB" formatted variable length records with embedded block and control words. The data encoding mode can be specified by STR, where STR can be ebcdic, ascii, or binary (or bin). The default is EBCDIC.

-ansi_db {STR}
specifies that the input tape file is ANSI-standard "DB" formatted variable length records with embedded record control words. The data encoding mode can be specified by STR, where STR can be ascii, ebcdic, or binary (or bin). The default is ASCII.

-output_description, -ods
allows the user to specify a standard Multics I/O attach description to receive the tape file data. User queries ask the user to input the attach description and the opening mode. Opening modes can be expressed in English terms or the appropriate abbreviation (e.g., sequential_output, sqo).

-extend
allows the user to concatenate the contents of several tape files into one output file. This argument has meaning only if the -count argument is also specified.

-nnl
allows escape from the read_file default of appending a new line character to the end of each physical record, when no other format specification is given.

-truncate N, -tc N
allows the user to truncate each physical record to a length of N characters.

-skip N
allows the user to skip N characters at the beginning of the physical tape record. This feature is primarily to allow a record or block control word to be skipped over while processing tapes of an unfamiliar format.

-logical_record_length N, -lrl N
allows the user to divide each physical tape record into several logical records of length N. Each logical record is written to the specified file with a new line character appended to the end. Logical records cannot span physical blocks.

read_tape_and_query

-convert STR, -conv STR
allows the user to convert the data format of each tape record, where STR can be one of the following:

ebcdic_to_ascii, ebcdic
converts input EBCDIC data to ASCII.

bcd_to_ascii, bcd
converts input BCD data to ASCII.

comp8_to_ascii, comp8
converts input comp8 (4 bit packed decimal) data to its equivalent ASCII representation.

Tape Positioning

When inspecting multi-file tape reels, you may find the action of various positioning requests confusing. The table below illustrates the starting and ending position when using various tape positioning requests:

Start Position	Operation	End Position
file 6, record 7	rewind	file 1, record 1
file 6, record 7	bof	file 6, record 1
file 6, record 7	bsf	file 5, record 1
file 6, record 7	fsf	file 7, record 1
file 6, record 7	bsr	file 6, record 6
file 6, record 7	fsr	file 6, record 8
file 6, record 7	bsf 8 (1)	file 1, record 1
file 6, record 7	bsr 10 (2)	file 6, record 1
file 6, record 1	read_file -count 3	file 9, record 1

note (1): This causes a rewind operation to take place, since the resultant file number would be less than 1.
note (2): This causes a bof operation to take place, since the resultant record number would be less than 1.

Examples

A typical example of a read_tape_and_query invocation follows, including the initial information displayed for a labeled tape.

```
read_tape_and_query usert1
Tape usert1,blk=2800 will be mounted with no write ring.
Tape usert1,blk=2800 mounted on drive tape_02 with no write ring.
Tape density is 1600 bpi
Tape usert1 is a labeled ANSI tape
Volume name recorded on tape label is USERT1
Setting tape dim to read in nine mode
First data file format:
 ANSI HDR2 label record. Next file format:
 Record format DB; Block length 4000; Record length 4000; Mode ASCII;
Positioning to beginning of physical tape file # 2, (logical file # 1)
-->
```

An example of the output produced by the list_tape_contents request for a labeled ANSI tape follows. Note the use of the -label and -long arguments:

```
--> list_tape_contents -label -long
Listing tape contents of tape volume usert1 in nine mode.
Starting at BOT (physical file # 1, record # 1)

Physical tape file # 1.

 ANSI VOL1 label record. Volume serial number USERT1
 ("VOL1USERT1                  MTF                                3")

 ANSI HDR1 label record. Data set ID RTQ.PL1
 ("HDR1RTQ.PL1    USERT1 00010001000100 80225 00000 000000MULTICS ANSI")

 ANSI HDR2 label record. Next file format:
 Record format DB; Block length 4000; Record length 4000; Mode ASCII;
 ("HDR2D0400004000                                  11 00")

End of physical tape file # 1, total records read - 3.

Physical tape file # 2.
 Logical tape file # 1.

End of physical tape file # 2, (logical tape file # 1),
total records read - 19.

Physical tape file # 3.

 ANSI EOF1 label record.
 ("EOF1RTQ.PL1    USERT1 00010001000100 80225 00000 000019MULTICS ANSI")

 ANSI EOF2 label record.
 ("EOF2D0400004000                                  11 00")

End of physical tape file # 3, total records read - 2.

Physical tape file # 4.

 ANSI HDR1 label record. Data set ID RD_TFILE.PL1
 ("HDR1RD_TFILE.PL1  USERT1 00010002000100 80225 00000 000000MULTICS ANSI")

 ANSI HDR2 label record. Next file format:
 Record format DB; Block length 4000; Record length 4000; Mode EBCDIC;
 ("HDR2D0400004000                                  12 00")

End of physical tape file # 4, total records read - 2.

Physical tape file # 5.
 Logical tape file # 2.

End of physical tape file # 5, (logical tape file # 2),
total records read - 1.

Physical tape file # 6.

 ANSI EOF1 label record.
 ("EOF1RD_TFILE.PL1  USERT1 00010002000100 80225 00000 000001MULTICS ANSI")

 ANSI EOF2 label record.
 ("EOF2D0400004000                                  12 00")
```

read_tape_and_query

```
End of physical tape file # 6, total records read - 2.

End of physical tape file # 7, total records read - 0.

Logical end of tape, positioning to BOT
-->
```

An example of the output produced by the list_tape_contents request for a labeled IBM tape follows:

```
--> list_tape_contents
Listing tape contents of tape volume usert2 in nine mode.
Starting at BOT (physical file # 1, record # 1)

Physical tape file # 1.

 IBM VOL1 label record. Volume serial number USERT2

 IBM HDR1 label record. Data set ID FILE1

 IBM HDR2 label record. Next file format:
 Record format VB; Block length 8192; Record length 8188;

End of physical tape file # 1, total records read - 3.

Physical tape file # 2.
 Logical tape file # 1.

  1 record: length = 73332 bits, 2037 words, 8148 nine bit bytes,
                     9166 eight bit bytes, 12222 six bit chars
  1 record: length = 73692 bits, 2047 words, 8188 nine bit bytes,
                     9211 eight bit bytes, 12282 six bit chars

End of physical tape file # 2, (logical tape file # 1),
total records read - 2.

Physical tape file # 3.

 IBM EOF1 label record.

 IBM EOF2 label record.

End of physical tape file # 3, total records read - 2.

End of physical tape file # 4, total records read - 0.

Logical end of tape, positioning to BOT
-->
```

An example of the output produced by the list_tape_contents request for a labeled GCOS tape follows:

```
--> list_tape_contents
Listing tape contents of tape volume usert3 in binary mode.
Starting at BOT (physical file # 1, record # 1)

Physical tape file # 1.

 GCOS BTL header label record; Tape reel # 70322.
```

read_tape_and_query

```
End of physical tape file # 1, total records read - 1.

Physical tape file # 2.
 Logical tape file # 1.

  46 records: length = 11124 bits, 309 words, 1236 nine bit bytes,
                       1390 eight bit bytes, 1854 six bit chars
   1 record: length = 3060 bits, 85 words, 340 nine bit bytes,
                      382 eight bit bytes, 510 six bit chars

End of physical tape file # 2, (logical tape file # 1),
total records read - 47.

Physical tape file # 3.

 GCOS "eof" label record. Block count of previous file 47.

End of physical tape file # 3, total records read - 1.

Physical tape file # 4.

 GCOS Partial header label record.

Logical end of tape, positioning to BOT
-->
```

An example of the read_file request, using the -count argument, to read in up to 99 files of a GCOS tape follows. Note that the label records are only identified and are not written to output files:

```
--> read_file -count 99 -gcos -output_file 3bt.ascii
Reading tape file # 1 in binary mode

 GCOS BTL header label record; Tape reel #  70322.
End of file after 1 record read from tape file # 1
Reading tape file # 2 in binary mode
Writing file 3bt.ascii.
End of file after 1 record read from tape file # 2
Reading tape file # 3 in binary mode

 GCOS "eof" label record. Block count of previous file 1.
End of file after 1 record read from tape file # 3
Reading tape file # 4 in binary mode

 GCOS header label record.
End of file after 1 record read from tape file # 4
Reading tape file # 5 in binary mode
Writing file 3bt.ascii.3.
End of file after 47 records read from tape file # 5
Reading tape file # 6 in binary mode

 GCOS "eof" label record. Block count of previous file 47.
End of file after 1 record read from tape file # 6
Reading tape file # 7 in binary mode

 GCOS header label record.
End of file after 1 record read from tape file # 7
Reading tape file # 8 in binary mode
Writing file 3bt.ascii.6.
End of file after 1 record read from tape file # 8
```

read_tape_and_query

```
Reading tape file # 9 in binary mode

 GCOS "eof" label record. Block count of previous file 1.
End of file after 1 record read from tape file # 9
Reading tape file # 10 in binary mode

 GCOS Partial header label record.

Logical end of tape at physical file # 10
-->.
```

tape_in

Name: tape_in

The tape_in command allows the user to transfer files between magnetic tape and the storage system. To accomplish a file transfer, the tape_in command accesses either the tape_ansi_ or the tape_ibm_ I/O module for the tape interface, and the vfile_ I/O module for the storage system interface. Unstructured format storage system files (for stream I/O) and sequential format storage system files (for record I/O) may be specified; 9-track ANSI standard labeled tapes, 9-track IBM standard labeled tapes, and any 9-track unlabeled tape structured according to OS standards may be read.

Usage

tape_in path {-control_args}

where:

1. path
   is the pathname of the control file governing the file transfer. If path does not end with the tcl suffix, it is assumed.

2. control_args
   can be chosen from the following:

   -severityN, -svN
   causes the tape_in compiler's error messages with severity less than N (where N is 0, 1, 2, 3, or 4) not to be written into the error_output I/O switch. The default value for i is 0. See "Error Diagnostics" below for further information on error reporting.

   -check, -ck
   performs only semantic checking on the Tape Control Language (TCL) control file. No tapes are mounted if this control argument is specified.

   -ring
   mounts volumes of the volume-set with write permit rings.

BASIC TCL CONTROL FILE

The control file that governs file transfer is actually a program, written by the user, in the Tape Control Language (TCL). The contents of this control file describe the file transfer(s) to take place. When the user issues the tape_in or tape_out command, the control file named in the command line by the path argument is compiled and if the compilation is successful, the generated code is interpreted to accomplish the desired file transfer(s). The same control file may be used with both the tape_in command (to read a file from tape into the storage system) and with the tape_out command (to write a file from the storage system onto tape).

The TCL control file consists of a list of statements of the form:

```
<keyword>:      <argument(s)>;
or
<keyword>;
```

These statements are combined to form file-groups and file-groups are combined to form volume-groups. A TCL control file consists of one or more volume-groups.

A file-group is a list of statements that define one tape to storage system file transfer. A file-group must begin with a File statement and must contain a path statement. In addition, it may contain one or more local statements. A file-group is terminated by a global statement, an End statement, or another File statement.

A volume-group is a series of statements that specify the file transfer(s) to be performed between the storage system and a particular tape volume-set. A volume-group must begin with a Volume statement, contain one or more file-groups, and terminate with an End statement. In addition, a volume-group may optionally contain one or more global statements, which apply to all the file-groups within the volume-group that follow the global statement.

All TCL control files must have at least four statements: a Volume statement, a File statement, a path statement, and an End statement; all other TCL statements are optional. The simplest control file has just these four statements, for example:

```
Volume:   012345;
File:   File1;
path:   >udd>Project_id>Person_id>demo;
End;
```

This example control file relies on TCL control file defaults, which are listed below under "Volume-Group Defaults." The file transfers possible with this sample control file are two: either writing tape file File1 from storage system file demo; or writing storage system file demo from tape file File1.

TCL CONTROL FILE STATEMENTS

Volume: <volid>;

The Volume statement specifies the tape volume to be used in file transfer. This statement causes a tape volume whose volume identifier is <volid> to be mounted on a 9-track drive. If <volid> contains any of the following characters, it must be enclosed in quotes.

1. any ASCII control character
2. : ; , or blank
3. the sequence /* or */
4. If <volid> itself contains a quote character, the quote must be doubled and the entire <volid> string enclosed in quotes.

tape_in

Some examples of Volume statements are:

```
Volume: 23;                    (mounts volume 23)
Volume: 001234;                (mounts volume 001234)
Volume: XJ56;                  (mounts volume XJ56)
Volume: "as"";56";             (mounts volume as";56)
Volume:  -00451;               (mounts volume -00451)
```

See the descriptions of tape_ansi_ and tape_ibm_ later in this manual for more details on volume specifications. Also, see "Multivolume Files" below for a discussion of multivolume volume-groups.

File: <fileid>;

The File statement specifies the tape file to be read or written. For output, <fileid> must be from one to 17 characters for ANSI labeled tapes and must be a valid DSNAME for IBM labeled tapes. A valid DSNAME is from one to eight characters long. The first character must be an alphabetic or national (@,$,#) character; the remaining characters can be any alphanumeric or national characters, a hyphen (-), or a plus zero (12-0 punch). For input, <fileid> may be an asterisk (*) for labeled tapes, if a tape file sequence number is also specified. For output with labeled tapes, <fileid> may not be an asterisk. <fileid> for IBM unlabeled tapes, which are discussed below, must be an asterisk. The File statement marks the beginning of any local attributes for a given tape file transfer.

path: <pathname>;

Associated with every File statement must be one path statement. The path statement specifies the pathname of the storage system file to be read or written. <pathname> may be either a relative or absolute pathname.

End;

Associated with every Volume statement must be an End statement, to mark the end of the TCL for that volume-group.

Global Statements

A global statement changes a volume-group default. The Tape and the Density global statements may appear only once in a volume-group and must preceed all file-groups. The Block, Expiration, Format, Mode, Record, and Storage global statements may appear any number of times within a volume-group. These statements apply to all subsequent file-groups within the volume-group.

Block: <blklen>;
The Block global statement specifies the tape file (maximum) physical block length, in bytes, to be used with subsequent file-groups. The <blklen> specification must be a decimal integer ≥18. For IBMSL, IBMNL, and IBMDOS formats, the maximum value is 32760 bytes. For ANSI formats, the maximum value is 99996 bytes. WARNING: <blklen> greater that 2048 does not comply with the ANSI standard for tapes.

Density: <den>;
The Density global statement indicates the density in which the volume is (to be) recorded. <den> must be either 800, 1600, 6250, 2, 3, or 4 (for IBM compatibility) to indicate 800, 1600 or 6250 bpi respectively. WARNING: the use of 1600 or 6250 bpi for ANSI interchange tapes is nonstandard. This global statement may appear only once within a volume-group or an error is indicated.

Expiration: <date>;
The Expiration global statement specifies the expiration date of files to be written (created). <date> is a string of a form acceptable to the convert_date_to_binary_ subroutine, for example "09/12/79". (See the convert_date_to_binary_ subroutine in the MPM Subroutines.) Because overwriting a file on a tape logically truncates the file set at the point of overwriting, the expiration date of a file must be earlier than or equal to the expiration date of the previous file (if any) on the tape; otherwise, an error is indicated. If an attempt is made to overwrite an unexpired file, the user is queried for explicit permission at the time of writing, unless the -force control argument is specified in the command line (only possible with tape_out).

Format: <form>;
The Format global statement specifies the tape record format to be used with subsequent file-groups. <form> must be either u, f, fb, d, db, s, or sb for ANSI tapes (using tape_ansi_ I/O module) and f, fb, u, v, vs, vb, or vbs for IBM tapes (using tape_ibm_ I/O module).

Mode: <mode>;
The Mode global statement specifies the tape mode and character code to be used with subsequent file-groups. <mode> may be either ascii or ebcdic for IBM tapes (using tape_ibm_ I/O module) and may be either ascii, ebcdic, or binary for ANSI tapes (using tape_ansi_ I/O module). WARNING: the use of ebcdic mode or binary mode is not standard for ANSI tapes. See "I/O Module Compatibility and Record Length Tables" below for a description of the interaction between a given combination of format, block, and record specification. Values must be carefully chosen to ensure desired results.

Record: <reclen>;
The Record global statement specifies the tape file (maximum) logical record length, in bytes, to be used with subsequent file-groups. <reclen> must be a decimal integer, such that 1!≤!<reclen>!≤!maximum!segment!size in bytes.

Storage: <structure>;
The Storage global statement states the internal (logical) structure of the storage system file(s) to be specified by subsequent file-groups. An unstructured file is referenced as a series of 9-bit bytes, commonly called lines; a sequential file is referenced as a sequence of records, each record being a string of 9-bit bytes. <structure> must be either unstructured or sequential. When an unstructured file is written into the storage system from a tape the NL character is appended as each line is written, unless the record already ends in a NL character, in which case nothing further is appended. When an unstructured file is written from the storage system to tape, the NL character is stripped off before writing the tape record. If a line of an unstructured file consists of just a NL character, it is written to tape as a zero length record. If the Storage global statement is omitted from a control file volume-group, the assumed storage system file format is unstructured. If a sequential file is referenced within that volume-group, the results are undefined and an error is indicated. Processing is terminated on that file in which the error is indicated.

Tape: <tape-type>;
The Tape global statement specifies the kind of tape that is processed. <tape-type> may be ibmsl for IBM standard labeled tape, ibmnl for IBM unlabeled tape, ibmdos for IBM DOS standard labeled tape, or ansi for ANSI standard labeled tape. The tape label processing is done automatically by the I/O module in use. This global statement may appear only once within a volume-group or an error is indicated.

Local Statements

A file-group may contain one or more local statements. A local statement overrides the volume-group defaults in effect at the time a file-group is evaluated. A local statement has no effect outside of the file-group in which it occurs and may appear anywhere within the file-group.

The block, expiration, format, mode, record and storage local statements operate exactly as do their global statement counterparts, except that they affect only the file-group in which they are contained.

generate;
The generate local statement causes the entire contents of a file on an ANSI tape to be replaced while retaining the structure of the file itself and incrementing the file generation number. The file to be modified is identified by the File statement, or by a combination of the File statement and the number statement.

modify;
The modify local statement causes the entire contents of a file on an ANSI or IBM labeled tape to be replaced while retaining the structure of the file itself. The file to be modified is identified by the File statement, or by a combination of the File statement and the number statement.

number: <number>;

The number statement specifies the file sequence number of the file to be used in the file transfer. <number> must be either an integer between 1 and 9999 inclusive, or the character "*". For input with labeled tapes, <number> = * is ignored unless * was specified for the <fileid> in the File statement. (In this case an error is indicated.) For output with labeled tapes, <number> = * appends the current file to the volume-set. If a tape volume has not yet been initialized, that is, if the first file to be written is the first file on that tape volume, <number> = * is considered a fatal error. Until a volume has been initialized, files cannot be appended to it. In this situation, either the number statement should be omitted or, if used, <number> must be equal to 1.

If the control file is to be used with the tape_in command, <number> specified in a number statement must correspond with a file on the specified tape volume-set. If both the <fileid> in the File statement and the <number> in the number statement are specified in the file-group, they must identify the same tape file; otherwise an error is indicated.

When reading unlabeled tapes, the number statement is required to identify the file to be read. When writing unlabeled tapes, the number statement is required to locate the tape position at which to write the file.

When the control file is to be used with the tape_out command for writing labeled tapes, the number statement is optional. If the number statement is given in a control file for use with the tape_out command, the file location specified in the number statement is the location where the file is written on the tape. Otherwise, with no number statement, the first file to be written in a volume-group is the first file position on the tape (for labeled tapes only). Subsequent files on that volume are appended after the first file.

replace: <fileid>;

If an existing tape file is to be replaced on an ANSI or IBM standard labeled tape and its name is known, the file to be overwritten is identified by <fileid> in the replace local statement and the new file to be written is identified by <fileid> in the File statement. If the file identified in the replace statement does not exist, an error is indicated.

storage_extend;

Normally when a user sets up a file-group to transfer a tape file to a storage system file, it is intended that a new file be created in the storage system. Should the user want to extend an already existing file in the storage system, the storage_extend local statement should be used in the TCL control file. If the storage system file to be extended does not exist, an error is indicated. If the storage_extend local statement exists in a control file used with tape_out, it is ignored.

tape_extend;

The tape_extend local statement allows new data records to be appended to an existing file on an ANSI or IBM standard labeled tape without in any way altering the previous contents of the tape file. The tape file to be extended is identified by the File statement or by the File statement and number local statement in combination. If the tape file to be extended does not exist on the tape, an error is indicated. Recorded in the labels of an ANSI or IBM labeled tape file is the version number. Initially, it is zero when the file is created. Every time a file is extended, its version number is incremented. The version number field is two digits and is reset to zero when the one-hundredth revision is made.

tape_in

CONTROL FILE COMMENTS

Comments may be inserted anywhere within the TCL program by surrounding the comment text with the comment delimiters. /* is the delimiter that begins a comment, and */ is the delimiter that terminates a comment.

VOLUME-GROUP DEFAULTS

Associated with a volume-group are a set of default characteristics. In the absence of overriding global statements or local statements, these defaults apply to all file-groups within the volume-group. If no tape-type is specified in the control file, ANSI standard labeled tape is assumed. If, however, a tape-type is specified (using a Tape statement), the volume-group defaults for that tape-type are in effect until overridden.

Tape-type ANSI or no Tape statement (this is the default)

1. density: 800 bpi
2. file expiration: immediate
3. storage system file format: unstructured
4. mode: ascii
5. tape file record format: variable length records, blocked
6. physical block length: 2048 characters (maximum)
7. logical record length: 2048 characters (maximum)

Tape-type ibmsl, ibmnl, or ibmdos

1. density: 1600 bpi
2. file expiration: immediate
3. storage system file format: unstructured
4. mode: ebcdic
5. tape file record format: variable length records, blocked
6. physical block length: 8192 characters (maximum)
7. logical record length: 8188 characters (maximum)

I/O MODULE COMPATIBILITY AND RECORD LENGTH TABLES

tape_ansi_

mode: ascii (default) ¦ binary ¦ ebcdic
block length: 18 < b < 99996 bytes (2048 default).
for output mode, block length must be divisible by 4.
density: d = 800 (default) ¦ 1600 ¦ 6250
file sequence number: 1 < n < 9999 or *
record length: 0 < r < 1044480
format: f = fb ¦ f ¦ db (default) ¦ d ¦ s ¦ sb ¦ u

tape_ibm_

mode: ascii ¦ ebcdic (default)
block length: 20 ≤ b ≤ 32760 bytes (8192 default)
for output mode, block length must be devisible by 4.
density: d = 800 ¦ 1600 (default) ¦ 6250
file sequence number: 1 < n < 9999 or *
record length: 0 < r < 1044480
format: f = fb ¦ f ¦ vb (default) ¦ v ¦ vbs ¦ u

Format	Record Length in bytes r	Block Length in bytes b
u	r is undefined	amrl < b < 99996 (tape_ansi_) amrl ≤ b ≤ 32760 (tape_ibm_)
f	r = amrl	b = r
fb	r = amrl	b must statisfy mod(b,r) = 0
d	amrl+4 < r < 99996	b = r
db	amrl+4 ≤ r ≤ 99996	b > r
s	amrl < r ≤ 1044480	18 ≤ b < 99996
sb	amrl ≤ r ≤ 1044480	18 ≤ b ≤ 99996
v	amrl+4 < r < 32756	b = r + 4
vb	amrl+4 ≤ r ≤ 32756	b > r + 4
vs	amrl < r ≤ 1044480	20 ≤ b < 32760
vbs	amrl ≤ r ≤ 1044480	20 ≤ b < 32760

Notes

amrl is the actual or maximum record length of a given record format, i.e., the actual or maximum number of characters that can be recorded in a logical record. The value of r is dependent on the choice of record format. For ANSI tapes, b must be an integer in the range of 18 < b < 99996. For IBM tapes, b must be an integer in the range of 20 < b < 32760. For ANSI tapes, in order to comply with the ANSI standard, b must be in the range of 18< b < 2048. For IBM tapes, the condition mod(b,4) = 0 must be satisfied. The TCL record statement should not be used for U-format file transfer.

ADDITIONAL OPTIONS AVAILABLE FOR THE TCL USER

A number of options are available to the user who wants to do more than the simple file transfer between a tape volume-set and the storage system. These features need not be of concern to most users, but for the user with specialized needs, these additional options are explained below.

Multivolume Files

Multivolume files are specified in a control file by a slightly more complicated Volume statement than shown above. The multiple <volid>s of such a volume-set are separated from one another by commas and are listed either in the order in which they became members of the volume-set, for input, or in the order in which they are candidates for volume-set membership, for output. The entire volume-set membership need not be specified in a Volume statement referencing a volume-set, but the first (possibly only) member must be mentioned. Up to 64 <volid>s may be specified in a single control file Volume statement.

Volume switching for multivolume files is handled automatically by the I/O modules. If sufficient volume-set members are given in the TCL control file, the volume switching is transparent to the user. If insufficient members of a volume-set are given or the membership is being developed, the user is queried during execution for names of additional volume-set members.

Sending Messages to the Operator

If it is necessary for the user to have a message displayed on the operator's console, the comment phrase can be included in the Volume statement. The comment text consists of the keyword -comment followed by the text of the message. Whenever the volume with the <volid> immediately preceding the comment phrase is to be mounted, the specified message is displayed on the operator's console. The message may be from 1 to 64 characters and must be a contiguous string with no embedded spaces or a quoted string with embedded quotes doubled. For example:

```
Volume:  060082 -comment "tape is Smith's" 060083 -comment tape_also_Smith's;
```

370/DOS Tapes

The tape_ibm_ I/O Module processes tapes created by or destined for IBM/DOS installations as well as tapes for IBM/OS installations. The Tape:!ibmdos; global statement is used in the TCL control file to specify that the tape files referenced by the given volume-group are destined for or have been produced by a IBM/DOS installation. The important difference between tape files created by OS and those created by DOS operating system is that the tape file structure attributes are not recorded in the tape labels under DOS. It is therefore necessary for all of the structure attributes of a DOS tape file, namely encoding mode, logical record format, logical record length, and block size to be specified in the TCL control file.

Unlabeled Tapes

The tape_ibm_ I/O Module supports processing of unlabeled tapes, provided that the tapes are structured according to the OS standard. DOS leading tape mark (LTM) unlabeled format tapes cannot be processed. The ibmnl specification in the Tape statement is mutually exclusive with any statement, global or local, which refers to labeled tapes: namely, the Expiration global statement and the expiration, generate, modify, replace, and tape_extend local statements. If any of these appear together within the same file-group, an error is indicated. When referencing unlabeled tape files in a given file-group, the argument of the File statement, <fileid>, must be specified by an asterisk, and the tape file desired must be specified by the number local statement.

ERROR DIAGNOSTICS

The error messages issued during tape_in and tape_out compilation are graded and have the form:

prefix error number, SEVERITY severity IN STATEMENT m OF LINE n
text of error message
SOURCE:
source statement in error

where n is the line number on which the described statement begins and m is a number identifying which statement in line n is in error. If line n contains only one statement, "STATEMENT m OF" is omitted from the error message.

The severity numbers produce one of the following prefixes:

severity	prefix	explanation
0	COMMENT	the error message is a comment.
1	WARNING	the error message warns that a possible error has been detected. However, the translation still proceeds.
2	ERROR	the error message warns that a probable error has been detected. However, the error is nonfatal, and the translation still proceeds.
3	FATAL ERROR	the error message warns that a fatal error has been detected. Processing of the input still continues to diagnose further errors, but no translation is performed.
4	TRANSLATOR ERROR	the error message warns that an error has been detected in the operation of the translator. No translation is performed.

tape_in

CONTROL FILE EXECUTION

When the TCL control file is being executed in response to the tape_in command, the volume named in each volume-group of the control file is mounted in turn without a write ring (unless the -ring control argument has been specified). If any output options appear in a control file being executed in response to the tape_in command, these statements are ignored. Then each file-group in that volume-group is processed resulting in one file transfer to the storage system per file-group.

FILE TRANSFER

File transfer is performed as follows. One logical record is read from the tape file, and as many characters as were read are written into the storage system file either as a line with newline (NL) character appended, if necessary, (unstructured case) or as one logical record in a sequential format file.

EXECUTION TIME DIAGNOSTICS

Any fatal error from an I/O module during execution of a control file causes the user to be queried as to whether or not he wishes to continue processing the other file-groups and volume-groups in the control file or whether to terminate processing of the control file. In the case of some correctable errors the user will be given the alternative of controlling the process. This alternative places the user at command level allowing resolution of the problem. When the user wishes to continue processing, the start command is used. Executing the release command will cause the tape_in command to be terminated.

CONTROL FILE EXAMPLES

Below are examples of typical control files. In the first example, the user wishes to load into the storage system, the contents of volume "2314dp" which contains a dump of a disk pack containing source and data.

The numbers at the left-hand side of the page in the examples below do not actually appear in the control file, but are included only for annotation reference.

tape_in

Example: sample1.tcl

```
!  tape_in sample1.tcl -ring

1        Volume: 2314dp;
2        /* Source Pack being loaded */
3        Tape: ibmsl;
4        Storage: unstructured;
5        Density: 800;
6        Format: fb;
7        Record: 80;
8        Block: 800;
9        File: FILEX;
10       path: <setup>data_entry>FILEX;
11       File: FILEXX;
12       path: <setup>data_entry>FILEXX;
13       File: FILEY;
14       path: <setup>data_entry>FILEY;
15       File: FILEYY;
16       path: <setup>data_entry>FILEYY;
17       File: FILEZ;
18       path: <setup>data_entry>FILEZ;
 .
 .
 .
58       File:  FILEZZ;
59       path: <setup>data_entry>FILEZZ;
60       End;
```

Annotations for sample1.tcl

1. mounts the volume 2314dp with a write ring.

2. comment.

3. specifies an IBM standard labeled tape.

4. files are created in unstructured format, ready for use in stream I/O. NL characters are appended as the file is written to disk. The mode is the default for the ibmsl tape-type, namely, ebcdic.

5. tape is recorded at 800 bpi.

6. all files on tape are in fixed block format unless stated otherwise. Possible record padding problems may be encountered.

7. all logical records are 80 characters unless stated otherwise (card image files).

8. all files blocked to 800 characters unless stated otherwise.

9. first file to be read from tape is named FILEX. It may be at any file location on the tape. The tape is automatically positioned to the file by name.

10. read tape file, FILEX, into storage system file named FILEX. The relative pathname, <setup>data_entry>FILEX, is expanded.

tape_in

11. continue reading files off the tape volume, one by one, into files in the storage system with the same name.

.
.
.

60. end of volume-group and end of control file.

Example: sample2.tcl Control File for Reading DOS tape

```
!  tape_in sample2.tcl

1      Volume:  042281 -comment "Please send tape to accounting";
2      Tape: ibmdos;
3      Density: 800;
4      Storage: unstructured;
5      Mode: ebcdic;
6      File: abc;
7      record: 80;
8      block:  800;
9      format: fb;
10     path: >udd>Example>Foo>fargo.pl1;
11     End;
```

Annotations for sample2.tcl

Note: Only selected statements in the control file are annotated here.

1. mount volume 042281 without a ring after printing comment message for operator.

2. read IBM DOS standard labeled tape.

4. read tape file into storage system as unstructured format files appending NL characters to each record from tape.

Example: sample3.tcl control file for Reading an Unlabeled Tape

```
!  tape_in sample3.tcl

1      Volume: 042381;
2      Tape: ibmnl;
3      Storage: sequential
4      File: *;
5      format:  vbs
6      number: 3;
7      path: >udd>Example>Foo>foobar.data;
8      End;
```

tape_in

Annotations for sample3.tcl

Note: Only selected statements in the control file are annotated here.

2. unlabeled tape is to be read. Files are unnamed. This statement must appear when processing unlabeled tapes.

4. <fileid> is specified by "*" for unnamed files.

6. the number statement must be present when processing unlabeled tapes. The third file on the tape is read.

The tape file record format is VBS, the tape file record length for VBS format is 1044480 bytes, and the tape file block length is 8192 bytes.

tape_out

Name: tape_out

The tape_out command allows the user to transfer files between the storage system and magnetic tape. To accomplish a file transfer, the tape_out command accesses either the tape_ansi_ or the tape_ibm_ I/O module for the tape interface, and the vfile_ I/O module for the storage system interface. Unstructured format storage system files (for stream I/O) and sequential format storage system files (for record I/O) may be specified; 9-track ANSI standard labeled tapes, 9-track IBM standard labeled tapes, and any 9-track unlabeled tape structured according to OS standard may be written.

Usage

tape_out path {-control_args}

where:

1. path
   is the pathname of the control file governing the file transfer. If pathname does not end with the tcl suffix, it is appended.

2. control_args
   can be chosen from the following:

   -severityN, -svN
   causes the tape_out compiler's error messages with severity less than N (where N is 0, 1, 2, 3, or 4) not to be written into the error_output I/O switch. The default value for i is 0. See "Error Diagnostics" in the tape_in command for further information on error reporting.

   -check, -ck
   specifies that only semantic checking be done on the Tape Control Language (TCL) control file. No tapes are mounted if this control is specified.

   -force, -fc
   specifies that the expiration date of a tape file to be overwritten is to be ignored. This control argument extends unconditional permission to overwrite a tape file, regardless of the file's "unexpired" status. This unconditional permission supresses any query made by the I/O module to inquire about tape file's expiration date.

   -ring
   mounts volumes of the volume-set with write permit rings (default).

TCL CONTROL FILE

The control file that governs file transfer for the tape_out command is written in the control file language described in the tape_in command.

ADDITIONAL OPTIONS AVAILABLE FOR THE TCL USER

A number of options are available to the user who wants to do more than the simple file transfer between storage and a tape volume-set. These features need not be of concern to most users, but for the user with specialized needs, these additional options are explained below.

Protecting Tape File From Accidental Overwriting

To protect tape files from being accidentally overwritten tape_ansi_ and tape_ibm_ include expiration dates in the tape labels they write. The expiration local statement or Expiration global statement can be used in the TCL source file. To overwrite or delete a tape file the current date must be later than the expiration date specified in the tape label. If this is not the case, the attempt to destroy the tape file will fail and an error will be indicated unless the -force control argument has been specified in the tape_out command line. In that case expiration date checking will not be done.

Special Outer Modes

Normally, when a user sets up a TCL control file file-group to write a storage system file onto a tape volume that is for use with the tape_out command it is intended that a new file be created on the tape volume. The TCL default output mode is create. This is the only output mode available for unlabeled tapes. For labeled tapes however, the TCL language offers four additional specialized output modes; they are generate, modify, replace, and tape_extend. The replace mode causes the tape file labels to be rewritten using specified and default file structure attributes. The tape_extend, modify, and generate local statements do not cause the tape file labels to be recomposed, so any file attributes specified in the file-group or volume-group that do not match those recorded in the tape labels, cause an error.

CONTROL FILE EXECUTION

When the TCL control file is being executed in response to the tape_out comand, the volume named in each volume-group of the control file is mounted in turn with a write ring. Then each file-group in that volume-group is processed resulting in one file transfer to the volume-set per file-group.

FILE TRANSFER

File transfer is performed as follows. Either a line or a record is read from the storage system file depending on whether the file is unstructured or structured. For unstructured format storage system files, a line read is a line from the file up to and including the first newline character (NL) encountered; for sequential format storage system files, a record read is one logical record of the file. The characters read from the storage system are then written on the tape as one logical record of the tape file.

tape_out

Under certain circumstances, tape records being written must be padded in accordance with a set of per-format padding rules. (For a discription of record and block padding for all formats, see the descriptions of tape_ansi_ and tape_ibm_.) Because of padding rules and treatment of newline characters when writing tape, a file that is written out to tape may not appear the same when read back in from tape. The following suggestions are offered:

1. to write character data (i.e., source files or text files) use the defaults; with tape_ansi_ use d, db, s, or sb format with the maximum block length, and the record length chosen so that the amrl (the actual or maximum record length of a given record format) is greater than the longest line in the storage system file. To avoid unwanted pad characters resulting from block padding, do not use f or fb format.

2. to write binary data with tape_ansi_, use the defaults with mode of binary or use s or sb format, with the maximum permissible block and/or record lengths and mode of binary.

3. to write character data with tape_ibm_, use vbs format with the maximum block length, and the record length chosen so that the amrl is greater than the longest line in the storage system file. (vb may cause one to three blanks to be appended to lines.)

4. when transfering sequential format files to tape, use a variable length record format (d, db, s, or sb with tape_ansi_ and v, vb, or vbs with tape_ibm_) to avoid unwanted padding characters being inserted into records. (vb may cause one to three blanks to be appended to lines.)

EXECUTION TIME DIAGNOSTICS

Any fatal error from an I/O module during execution of a control file causes the user to be queried as to whether or not he wishes to continue processing the other file-groups and volume-groups in the control file or whether to terminate processing of the control file. In the case of some correctable errors the user will be given the alternative of "controlling the process." This alternative places the user at command level allowing resolution of the problem. When the user wishes to continue processing, the start command is used. Executing the release command will cause the tape_out command to be terminated.

CONTROL FILE EXAMPLES

Below are examples of typical control files. In the first example, the user wishes to produce two tapes, one for the Multics system, the other for an Os installation. The Multics tape contains the source code of user subsystem SUBSYS, as well as its object code. The OS tape contains only the source code.

Example: sample1.tcl

```
!  tape_out sample1.tcl

1          Volume: 001234;
2          /* Dump source in DB and object in SB format */
3          File: FILE_1;
4          path: SUBSYS.pl1;
5          File:  FILE_2;
6          mode: binary;
7          path: <object>SUBSYS;
8          format: SB;
9          End;
10         Volume: DFG054;
11         /* append source to tape */
12         Tape: ibmsl;
13         File:  TESTSAVE;
14         format: VBS;
15         block: 4096;
16         path: SUBSYS.pl1;
17         number: 3;
18         End;
```

Annotations for sample1.tcl

1. mounts volume 001234 with a ring. The volume defaults are set to ANSI standard labeled tape-type, 800 bpi density, ASCII encoding mode, DB record format, block length = 2048, and record length = 2048.

2. is a comment in the control file. Since the storage statement is missing, the default storage system file format is set to transfer unstructured files.

3. since there is no number statement, the default positions the tape so that FILE_1 is created as a new file at the first file position on the tape volume.

4. specifies the pathname of the storage system file to be written to tape. Since the file-group contains no local statements, the file is written according to the current volume defaults.

5. positions the tape so that the file to be written is appended at file position two on the tape volume.

6. specifies that the file is to be written in binary encoding mode.

7. specifies the pathname of the storage system file to be written to tape.

8. specifies that the file is to be written in SB format. Notice that the block length is the current volume default block length (2048) and the record length is the current volume default record length (2048).

9. signifies end of volume-group. The I/O switch is closed and detached. The volume-set is taken down and the drive is released.

10. mounts volume DFG054 with a ring.

tape_out

11. is a comment. Storage format is still unstructured.

12. changes tape-type to IBM standard labeled; changes the volume-group defaults to those associated with ibmsl: 1600 bpi, ebcdic, VB format, block length = 8192, and record length = 8188.

13. specifies name of file to be written onto tape. Notice that the underscore (_) cannot appear in an IBM file name whereas it can appear in an ANSI file name.

14. changes the record format to VBS. A spanned record format transfering a sequential file is needed, so that unwanted block padding is not inserted into the file as it is transferred. The default record length for VBS format is 1044480 bytes.

15. changes the block length to 4096.

16. specifies the pathname of the storage system file to be written.

17. This number statement is required to make sure the file is appended to an already existing tape volume. Without this number statement, the file would be created as the first file on the tape volume, overwriting any existing files. If files one and two do not exist, an error is indicated, but if these files do exist, the file is written at file position three on the tape volume.

18. rewinds and takes down the volume since no more file-groups in the control file reference the current tape volume.

Example: sample2.tcl

```
! tape_out sample2.tcl -fc

1     Volume: 070067 -comment in_slot_1000, 070068;
2     Tape: ansi;
3     File: BIG_LISTING;
4     replace: FILE_20;
5     number: 20;
6     expiration: 2weeks;
7     format: db;
8     block: 2048;
9     record: 133;
10    path: >udd>Example>Mega>test.list
11    End;
```

Annotations for sample2.tcl

1. The first member of the volume-set, 070067, is mounted without a ring, displaying the message "in_slot_10000" on the operator's console. Later if necessary, the volume-set member 070068 may be mounted to continue writing a large listing file. A message appears upon mounting the second member of the volume-set.

2. writing an ANSI standard tape.

tape_out

3. tape file named BIG_LISTING, into which the storage system file is to be written.

4. is to replace tape file named FILE_20.

5. by the number statement FILE_20 is the 20th file on the current volume-set. As no density statement is included in the control file, the default for tape_ansi_, 800 bpi, is used. Upon execution of the control file, the tape is positioned at the 20th file automatically, providing 20 files exist on the tape. As no Storage statement is present in the control file, the default storage system format is unstructured, and as the files are written to tape, the NL character is stripped.

6. The file, BIG_LISTING, is protected against accidental overwriting for two weeks, meaning that if the user attempts to overwrite the file within that time, he is first queried for permission to do so. The -force control argument in the command line inhibits a query for permission to overwrite FILE_20, in case it has not yet expired.

7. BIG_LISTING is recorded in variable length blocked record format. Mode is the default for tape_ansi_, namely ascii.

8. Block length is maximum allowed for ANSI interchange standard, 2048.

9. record length is 133.

10. the listing file is transferred from test.list in the storage system.

11. signifies termination of volume-group and of control file.

If, after putting his listing file out onto tape, the user wishes to delete the on-line listing, and at a later time, read the listing back from tape into storage, he might type the command line:

```
tape_in sample2.tcl
```

The output statements in the control file, namely the replace local statement and the expiration local statement are ignored on input.

SECTION 5

I/O MODULES

This section contains descriptions of Multics I/O modules, presented in alphabetic order. Each description contains the name of the I/O module, discusses its purpose, and describes the attach description and the operations supported by the I/O module. Notes and examples are included when deemed necessary for clarity.

The I/O modules described in this section and their functions are:

ntape_	supports I/O from/to magnetic tape file
rdisk_	supports I/O from/to removable disk packs
tape_ansi_	implements the processing of magnetic tape files according to standards proposed by the American National Standards Institute (ANSI)
tape_ibm_	implements the processing of magnetic tape files according to standards established by IBM
tape_mult_	supports I/O from/to Multics standard tape
tape_nstd_	supports I/O from/to tapes in nonstandard or unknown formats

ntape_

Name: ntape_

The ntape_ I/O module supports I/O on files on magnetic tape.

Entry points in the 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 and a discussion of files.

Attach Description

The attach description has the following form:

```
ntape_ reel_num -raw {-control_args}
```

where:

1. reel_num
   is the tape reel number. If the tape is 7-track, reel_num must contain ",7track". If the tape is 9-track, reel_num may contain ",9track" (if it contains neither, 9-track is assumed).

2. -raw
   indicates that each physical record (block) on the tape represents one logical record.

3. control_args
   may be one of the following arguments:

   -write
   means that the tape is to be mounted with a write ring. This argument must occur if the I/O switch is to be opened for output or input/output.

   -extend
   specifies extension of the file if it already exists on the tape.

Opening

The opening modes supported are sequential_input, sequential_output, and sequential_input_output. If an I/O switch attached via the ntape_ I/O module is to be opened for output or input_output, the -write argument must occur in the attach description.

Control Operation

This I/O module does not support the control operation.

Modes Operation

This I/O module does not support the modes operation.

Notes

On input, the logical record contains m=4*ceil(n/36) bytes, where n is the number of data bits in the physical record. The first n bits of the input record are the data bits, the last (9*m-n) bits are 0's.

On output, the physical record contains n=k*ceil((36*ceil(m/4))/k) data bits, where k+1 is the number of tracks on the tape, and m is the length of the logical record in characters. The first 9*m data bits of the physical record contain the bits of the logical record (i.e., the output buffer). The last (n-9*m) bits of the physical record are 0's.

This I/O module assumes that there is only one physical file on the tape. It is not possible to position the tape after a tape mark. The tape_nstd_ I/O module should be used to read nonstandard formatted tapes containing more than one tape mark.

rdisk_

Name: rdisk_

The rdisk_ I/O module supports I/O from/to disk packs. Sequential and indexed file types are supported.

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

Attach Description

The attach description has the following form:

```
rdisk_ device_id pack_id {-control_args}
```

where:

1. device_id
   is a character string identifying the type number of the required disk device. The supported disk devices are listed in the table below, along with the character string to use for device_id:

device_id Character String	Device Type
d181	DSU181
d190	DSU190
d191 or d400	DSU190/MSU0400 with the high-efficiency format (40 sectors/track)
d451	MSU0451
d500	MSU0500
d501	MSU0501

2. pack_id
   is a character string identifying the disk pack to be mounted.

3. control_args
   may be chosen from the following and may occur only once:

   -write
   indicates that the disk pack may be written on. If omitted, the operator is instructed to mount the pack with the PROTECT button pressed so that writing is inhibited.

   -size N
   indicates that the value of N is to override the value of the buff_len parameter as a record size limit for the read_record operation. (See "Notes" below.)

   -sys
   indicates that the attachment is being made by a system process and that a disk drive reserved for system functions is to be assigned.

The attachment causes the specified disk pack to be mounted on a drive of the specified type.

Opening

The following opening modes are supported:

sequential_input
sequential_output
sequential_update
direct_input
direct_update

Notice that if the opening mode is of the output or update type, the attach description must include the -write control argument so that the operator does not press the PROTECT button when the pack is mounted.

Delete Record Operation

This operation is not supported.

Read Length Operation

This operation is not supported.

Position Operation

This operation is supported for only the sequential_input and sequential_update opening modes. The type and quantity values are interpreted as follows:

type	quantity	action
-1	--	position to the beginning of the file.
+1	--	position to the end of the file.
0	N	skip N sectors (forward if N > 0; backward if N < 0).
2	N	position to sector N.

Read Record Operation

If the amount of data to be read does not terminate on a sector boundary, the excess portion of the last sector is discarded. A code of 0 is returned in this case. (See "Notes" below.) This operation is not supported for the sequential_output opening mode.

rdisk_

Rewrite Record Operation

If the amount of data to be written does not terminate on a sector boundary, the remaining portion of the last sector is filled with spaces in sequential modes and binary zeros in direct modes. A code of 0 is returned in this case. (See "Notes" below.) This operation is supported for only the update opening modes.

Seek Key Operation

This operation returns a status code of 0 for any key that is a valid sector number. The record length returned is always 256 (current physical sector size in characters) for any valid key. The specified key must be a character string that could have been produced by editing through a PL/I picture of "(8)9". (See "Notes" below.) This operation is supported for only the direct opening modes.

Control Operation

The following orders are supported when the I/O switch is open, except for getbounds, which is supported while the switch is attached.

changepack
: causes the current pack to be dismounted and another pack to be mounted in its place. The info_ptr should point to a varying character string (maximum of 32 characters) containing the identifier of the pack to be mounted. This operation is not allowed for MSU0500 or MSU0501 devices.

device_info
: causes information pertaining to the attached disk device to be returned to the user. The info_ptr should point to a structure of the following form:

```
dcl 1 device_info_table aligned,
    2 subsystem_name    char (4),
    2 device_name       char (8),
    2 sect_per_dev      fixed bin (35),
    2 cyl_per_dev       fixed bin,
    2 sect_per_cyl      fixed bin,
    2 sect_per_track    fixed bin,
    2 num_label_sect    fixed bin,
    2 num_alt_sect      fixed bin,
    2 sect_size         fixed bin (12);
```

where:

1. subsystem_name
   is the name of the disk subsystem in use (i.e., D191).

2. device_name
   is the name of the disk device in use (i.e., dska_04).

3. sect_per_dev
   is the total number of non-T&D sectors on the disk pack.

4. cyl_per_dev
   is the total number of non-T&D cylinders on the disk pack.

5. sect_per_cyl
   is the number of data sectors on each cylinder of a disk pack.

6. sect_per_track
   is the number of data sectors on each track.

7. num_label_sect
   is the number of data sectors to reserve for label information.

8. num_alt_sect
   is the number of data sectors to reserve for alternate track area.

9. sect_size
   is the number of 36-bit words in each data sector.

format_trk
causes a format track command to be issued to the track that was indicated by a preceding seek_key operation. This operation is not allowed for MUS0500 or MSU0501 devices. The info_ptr should point to a user supplied structure of the following form:

```
dcl 1 format_trk_info aligned,
   (2 hz              bit (2),
    2 ti              bit (2),
    2 adcyl           fixed bin (16),
    2 adhd            fixed bin (16)) unaligned;
```

where:

1. hz
   is a bit pattern indicating the state of the header bypass switch. The hz bits are defined as follows:

h z	bit pattern meaning
0 0	format home address and all data records
0 1	verify home address and record one, format home address and all data records
1 0	skip home address, format all data records
1 1	verify home address and data record one, skip home address and format all data records

rdisk_

2. ti
is a bit pattern indicating the state of the track indicator bits. The ti bits are defined as follows:

t i	bit pattern meaning
0 0	format track good
0 1	format track alternate
1 0	format track defective with alternate track assigned
1 1	format track defective with no alternate track assigned

3. adcyl and adhd
are the alternate or defective cylinder and head numbers used when the track indicator bits equal "01"b or "10"b. These two fields are defined as follows:

If the track indicator bits are set to "01"b (alternate track), then adcyl and adhd should be equal to the defective cylinder and head number for which the alternate track is being formatted.

If the track indicator bits are set to "10"b (defective with alternate assigned), then adcyl and adhd should be equal to the cylinder and head number of the alternate track. This operation is not allowed for MUS0500 or MSU0501 devices.

getbounds
causes the lowest and highest sector numbers accessible by the caller under the current modes to be returned. The info_ptr should point to a structure of the following form:

```
dcl 1 bounds,
    2 low      fixed bin(35),
    2 high     fixed bin(35);
```

rd_trk_header
causes a read track header command to be issued to the track that was indicated by a preceding seek_key operation. This operation is not allowed for MUS0500 or MSU0501 devices. The raw track header information is passed to the user in a structure (pointed to by info_ptr) of the following form:

```
dcl 1 trk_header_info aligned,
   (2 ha_cyl           bit (16),
    2 ha_head          bit (16),
    2 pad1             bit (2),
    2 ha_ti            bit (2),
    2 pad2             bit (10),
    2 rcd_0_ti         bit (2),
    2 rcd_0_cyl        bit (16),
    2 rcd_0_head       bit (16),
    2 rcd_0_rn         bit (8),
    2 pad3             bit (24),
    2 rcd_0_data (8),  bit (8),
    2 pad4             bit (4)) unaligned;
```

where:

1. ha_cyl
is the cylinder number read from the track home address.

2. ha_head
   is the head number read from the track home address.

3. ha_ti
   is the track indicator bits (defined above in the format_trk order) read from the track home address.

4. rcd_0_ti
   is the track indicator bits read from record zero. If the ha_ti bits indicate "10"b, then rcd_0_ti should equal "01"b for alternate track. If ha_ti indicates "01"b, then rcd_0_ti should equal "10"b for defective track. Otherwise rcd_0_ti will equal ha_ti.

5. rcd_0_cyl and rcd_0_head
   are the cylinder and head number read from record zero. If ha_ti indicates "10"b, then rcd_0_cyl and rcd_0_head equal the cylinder and head number of the alternate track. If ha_ti indicates "01"b, then rcd_0_cyl and rcd_0_head contain the cylinder and head number of the defective track. Otherwise, rcd_0_cyl and rcd_0_head equal ha_cyl and ha_head.

6. rcd_0_rn
   is the record number for record zero (normally equal to zero).

7. rcd_0_data
   is the eight data bytes in record zero (not a normal data record) and is normally equal to zero.

8. padn
   are unused bits that are returned as "0"b.

setsize
causes the value of the record size override setting to be reset. The info_ptr should point to an aligned fixed binary(35) quantity containing the new override value.

Modes Operation

The modes operation is supported when the I/O switch is attached. The recognized modes are listed below. Each mode has a complement indicated by the circumflex character (^) that turns the mode off.

label, ^label
specifies that a system-defined number of sectors at the beginning of the pack are reserved for a pack label, and that a seek_key or position operation is to treat any key or position within this area as an invalid key. (The default is on.)

raw, ^raw
specifies that the entire disk pack is available to the user, including the T&D cylinder (the last cylinder on the disk pack). (The default is off.)

alttrk, ^alttrk
specifies that the pack has been formatted with the assignment of alternate tracks, so that a system-defined number of sectors at the end of the pack are reserved for an alternate track area. Therefore, a seek_key or position operation is to treat any key within that area as an invalid key. (The default is off.) This mode cannot be enabled for a MSU0500 or MSU0501 disk.

wrtcmp, ^wrtcmp
specifies that the write-and-compare instruction, rather than the write instruction, is used for the rewrite_record operation. This causes all data written to be read back and compared to the data as it was prior to being written. This mode should be used with discretion, since it doubles the data transfer time of every write. (The default is off.)

Write Record Operation

If the amount of data to be written does not terminate on a sector boundary, the remaining portion of the last sector is filled with spaces. A code of 0 is returned in this case. (See "Notes" below.) This operation is supported for only the sequential_output opening mode. A series of writes will write successive records.

Closing

The closing has no effect on the physical device. For the sequential_output opening mode, the effect is as if an end-of-file flag is placed just beyond the end of the available disk area.

Detaching

The detachment causes the disk pack to be detached from the users process.

Notes

This I/O module is a very elementary, physical-device-oriented I/O facility, providing the basic user-level interface to a disk device. All operations are performed through calls to various I/O interfacer (IOI) mechanisms and resource control package (RCP) entries. Certain conditions must be satisfied before a user process can make use of this facility:

1. The system must be configured with one or more disk drives available as I/O disks.

2. The user must have access to assign the disk drive with RCP, access to the IOI gates, and access to the "acs" segment (e.g., >sc1>rcp>dskb_18.acs) that is used by the site to control access to the disk drive.

For input and update opening modes, the file occupies the entire available disk area (see the getbounds control order). For the sequential_output opening mode, the file is considered to be empty. That is, an open followed by a write records data in the first sector of the available disk area.

For direct opening modes, the entire disk pack is treated as an indexed file, with keys interpreted literally as physical sector numbers. Hence, the only allowable keys are those that can be converted into fixed binary integers that fall within the range of valid sector numbers for the given disk device under the current modes, as returned by the getbounds control operation.

For the sequential_input and sequential_update opening modes, if an attempt is made to read beyond the end of the user-accessible area, the code error_table_$end_of_info is returned. For all other opening modes, if an attempt is made to read or write beyond the end of the user-accessible area on disk, the code error_table_$device_end is returned. If a defective track is encountered or if any other unrecoverable data transmission error is encountered, the code error_table_$device_parity is returned.

The record length is specified through the buff_len parameter in the read_record operation, and through the rec_len parameter for the write and rewrite operations, unless overridden by a -size control argument in the attach description, or by a setsize control order.

The following items must be considered when using this I/O module with language input/output:

1. Device Attachment and File Opening:

   a. PL/I: A file can be attached to a disk pack in PL/I by specifying the appropriate attach description in the title option of an open statement. After opening, the desired modes should be set and the current sector bounds should be obtained through direct calls to iox_$find_iocb, iox_$modes, and iox_$control. These iox_ subroutine entry points are described in the MPM Subroutines.

   b. FORTRAN: It is not possible to attach a file to a disk pack within FORTRAN. Here, the attachment must be made external to the FORTRAN program, e.g., through the io_call command (described in the MPM Commands) or through use of a PL/I subroutine. FORTRAN automatically opens the file with the appropriate attributes. Also, it is impossible to set modes or obtain sector bounds from within FORTRAN. This should be done through use of a PL/I subroutine prior to the first FORTRAN reference to the file.

2. Input:

   a. PL/I: The input record length (buff_len) is determined by the size of the variable specified in the into option.

rdisk_

For the sequential_input and sequential_update opening modes, use the PL/I read statement with the into option to read data. Use the ignore option to skip forward within the file. An open statement followed by a read statement will read in the first record. Successive reads will obtain successive records.

For the direct_input opening mode, use the PL/I read statement with the into and key options. The set option should not be used. The key should be a character string containing the character representation of the desired sector number.

The PL/I get statement can be used with the sequential_input opening mode if the record_stream_ I/O module is referenced in the attach description of the open statement.

b. FORTRAN: In FORTRAN, buff_len has no relationship to input variable size. Hence, the -size control argument must be specified in the attach description if the disk pack is to be read through FORTRAN. The size should be set to the length of the longest expected record.

For the sequential_input opening mode, use the unformatted sequential read statement.

For the direct_input opening mode, use the unformatted keyed version of the FORTRAN read statement. The key must be an integer, whose value is the desired sector number.

3. Output:

a. PL/I: The size of the variable referenced in the from option determines the length of the record written to disk.

For the sequential_output opening mode, use the write statement with the from option. An open statement followed by a write statement will start writing at the beginning of the available area on the disk pack.

For the sequential_update opening mode, use the rewrite statement with the from option. A previous read statement must have been used to designate which record will be updated.

For the direct_update opening mode, use the rewrite statement with the from and key options. The key should be a character string containing the character representation of the desired sector number.

The PL/I put statement can be used with the sequential_output opening mode if the record_stream_ I/O module is referenced in the attach description of the open statement.

b. FORTRAN: The size of the output record is determined by the amount of data specified in the write list.

For the sequential_output opening mode, use the unformatted sequential write version of the FORTRAN write statement.

For the direct_update opening mode, use the unformatted keyed version of the write statement. The key should be a character string containing the character representation of the desired sector number.

Control Operations From Command Level

All control operations may be performed from the io_call command, as follows:

```
io_call control switch order_arg
```

where:

1. switch
   is the name of the I/O switch.

2. order_arg
   must be one of the following:

   ```
   changepack newpack
   setsize newsize
   getbounds
   ```

   where:

   newpack is the name of the new pack to be mounted.

   newsize is the new record size in words.

tape_ansi_

Name: tape_ansi_

The tape_ansi_ I/O module implements the processing of magnetic tape files according to Draft Proposed Revision X3L5/419T of the American National Standards Institute's ANSI X3.27-1969, "Magnetic Tape Labels and File Structure for Information Interchange". This document is referred to below as the DPSR (Draft Proposed Standard Revision). In addition, the I/O module provides a number of features that are extensions to, but outside of, the DPSR. Using these features may produce a nonstandard file, unsuitable for interchange purposes.

Entries in the 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. They represent a simplification and combination of the exact and complete set of definitions found in the DPSR.

record
: related information treated as a unit of information.

block
: a collection of characters written or read 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 records pertaining to a single subject. A file may be recorded on all or part of a volume, or on more than one volume.

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.

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

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

Attach Description

The attach description has the following form:

tape_ansi_ vn1 vn2 ... vnN {-control_args}

where:

1. vni
   is a volume specification. A maximum of 64 volumes may be specified. In the simplest (and typical) case, a volume specification is a volume name, that must be six characters or less in length. If a volume name is less than six characters and entirely numeric, it is padded on the left with 0's. If a volume name is less than six characters and not entirely numeric, it is padded on the right with blanks. Occasionally, keywords must be used with the volume name. For a discussion of volume names and keywords see "Volume Specification" below.

   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.

   -block b, -bk b
   specifies the block length in characters, where the value of b is dependent upon the value of r specified in the -record control argument. (See "Creating a File" below.)

   -clear, -cl
   specifies that internal information on a file-set which the I/O module retains from previous attachments is to be deleted. This control argument can be used when it is desired to change attributes of a file-set which are maintained across attachments for a given process, e.g. density or label standard. For the initial attachment to a file-set in a given process, this control argument has no effect.

   -create, -cr
   specifies that a new file is to be created. (See "Creating a File" below.)

   -density N, -den N
   specifies the density at which the file-set is recorded, where N can be 800, 1600, or 6250 bits per inch. (See "File Set Density" below.)

-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 ≤ N ≤ 63. (See "Multiple Devices" below.)

-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. (See "File Expiration" below.)

-extend, -ext
specifies extension of an existing file. (See "Extending a File" below.)

-force, -fc
specifies that the expiration date of the file being overwritten is to be ignored. (See "File Expiration" below.)

-format f, -fmt f
specifies the record format, where f is a format code. (See "Creating a File" below for a list of format codes.)

-generate, -gen
specifies generation of an existing file. (See "Generating a File" below.)

-mode STR, -md STR
specifies the encoding mode used to record the file data, where STR is the string ascii, ebcdic, or binary. The default is ascii. (See "Encoding Mode" below.)

-modify, -mod
specifies modification of an existing file. (See "Modifying a File" below.)

-name STR, -nm STR
specifies the file identifier of the file where STR is from 1 to 17 characters. (See "Creating a File" below.)

-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 ≤ N ≤ 9999. (See "Creating a File" below.)

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

-replace STR, -rpl STR
specifies the file identifier of the file to be replaced, where STR must be from 1 to 17 characters. If no file with file identifier STR exists, an error is indicated. (See "Creating a File" below.)

-retain STR, -ret STR
specifies retention of resources across attachments, where STR specifies the detach-time resource disposition. (See "Resource Disposition" below.)

-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.)

The following sections define each control argument in the contexts that it can be used. For a complete list of the attach control arguments, see "Attach Control Arguments" below.

Creating A File

When a file is created, an entirely new entity is added to the file set. There are two modes of creation: append and replace. In append mode, the new file is added to the file set immediately following the last (or only) file in the set. The process of appending does not alter the previous contents of the file set. In replace mode, the new file is added by replacing (overwriting) an existing file. The replacement process logically truncates the file set at the point of replacement, destroying all files (if any) that follow consecutively from that point.

The -create and -name control arguments are required to create a file, where STR is the file identifier. No two files in a file set can have the same file identifier. If the act of creation would cause a duplication, an error is indicated.

If no file having file identifier STR exists in the file set, the new file is appended to the file set; otherwise, the new file replaces the old file of the same name.

If the user wishes to explicitly specify creation by replacement, the particular file to be replaced must be identified. Associated with every file is a name (file identifier) and a number (file sequence number.) Either is sufficient to uniquely identify a particular file in the file set. The -number N and -replace STR control arguments, either separately or in conjunction, are used to specify the file to be replaced. If used together, they must both identify the same file; otherwise, an error is indicated.

This page intentionally left blank.

When the -number N control argument is specified, if N is less than or equal to the sequence number of the last file in the file set, the created file replaces the file having sequence number N. If N is one greater than the sequence number of the last file in the file set, the created file is appended to the file set. If N is any other value, an error is indicated. When creating the first file of an entirely new file set, the -number 1 control argument must be explicitly specified. (See "Volume Initialization" below.)

The -format f, -record r and -block b control arguments are used to specify the internal structure of the file to be created. They are collectively known as structure attribute control arguments.

tape_ansi_

When the -format f control argument is used, f must be one of the following format codes, chosen according to the nature of the data to be recorded. (For a detailed description of the various record formats, see "Record Formats" below.)

fb for fixed-length records, blocked. Used when every record has the same length, not in excess of 99996 characters.

db for variable length records, blocked. Used when records are of varying lengths, the longest not in excess of 99992 characters.

sb for spanned records, blocked. Used when the record length is fixed and in excess of 99996 characters, or variable and in excess of 99992 characters. In either case, the record length cannot exceed 1,044,480 characters.

f for fixed-length records, unblocked.

d for variable-length records, unblocked.

s for spanned records, unblocked.

u for undefined records (records undefined in format). Each block is treated as a single record, and a block may contain a maximum of 99996 characters.

NOTE: THE USE OF UNDEFINED RECORDS IS A NONSTANDARD FEATURE.

Records recorded using U format may be irreversibly modified; therefore, the use of U format is strongly discouraged. (See "Block Padding" below.)

Unblocked means that each block contains only one record (f, d) or record segment (s). Blocked means that each block contains as many records (fb, db) or record segments (sb) as possible. The actual number of records/block is either fixed (fb), depending upon the block length and record length, or variable (db, sb), depending upon the block length, record length, and actual records. Because of their relative inefficiency, the use of unblocked formats is discouraged.

When the -record r control argument is used, the value of r is dependent upon the choice of record format. In the following list, amrl is the actual or maximum record length.

f = fb ¦ f: r = amrl
f = db ¦ d: amrl + 4 ≤ r ≤ 99996
f = sb ¦ s: amrl ≤ r ≤ 1044480
f = u: r is undefined
(the -record control argument should not be used.)

When the -block b control argument is used, the value of b is dependent upon the value of r. When the block length is not constrained to a particular value, the largest possible block length should be used.

f = fb:	b must satisfy mod (b,r) = 0
f = f:	b = r
f = db:	b > r
f = d:	b = r.
f = sb | s:	18 < b < 99996
f = u:	amrI ≤ b ≤ 99996

In every case, b must be an integer in the range 18 ≤ b ≤ 8192.

NOTE: THE USE OF A BLOCK LENGTH IN EXCESS OF 2048 CHARACTERS IS A NONSTANDARD FEATURE.

Because the structure attribute control arguments are extremely interdependent, care must be taken to ensure that specified values are consistent.

Reading A File

The attach description needed to read a file is less complex than the description used to create it. When a file is created, the structure attributes specified in the attach description are recorded in the file's header and trailer labels. These labels, which precede and follow each file section, also contain the file name, sequence number, block count, etc. When a file is subsequently read, all this information is extracted from the labels. Therefore, the attach description need only identify the file to be read; no other control arguments are necessary.

The file can be identified using the -name STR control argument, the -number N control argument, or both in combination. If the -name STR is used, a file with the specified file identifier must exist in the file set; otherwise, an error is indicated. If the -number control argument is used, a file with the specified file sequence number must exist in the file set; otherwise, an error is indicated. If the -name STR and -number N control arguments are used together, they must both refer to the same file; otherwise, an error is indicated.

Output Operations On Existing Files

Three output operations can be performed on an already existing file: extension, modification, and generation. As their functions are significantly different, they are described separately below. They do, however, share a common characteristic. Like the replace mode of creation, an output operation on an existing file logically truncates the file set at the point of operation, destroying all files (if any) that follow consecutively from that point.

tape_ansi_

Extending A File

File extension is the process of adding records to a file without in any way altering the previous contents of the file.

Because all the information regarding structure, length, etc. can be obtained from the file labels, the attach description need only specify that an extend operation is to be performed on a particular file. The previous contents of the file remain unchanged; new data records are appended at the end of the file. If the file to be extended does not exist, an error is indicated.

The file to be extended is identified using the -name STR control argument, the -number N control argument, or both in combination. The same rules apply as for reading a file. (See "Reading a File" above.)

Recorded in the labels that bracket every file section is a version number, initially set to 0 when the file is created. The version number is used to differentiate between data that have been produced by repeated processing operations (such as extension). Every time a file is extended, the version number in its trailer labels is incremented by 1. When the version number reaches 99, the next increment resets it to 0.

The user may specify any or all of the structure attribute control arguments when extending a file. The specified control arguments are compared with their recorded counterparts; if a discrepancy is found, an error is indicated.

Modifying A File

It is occasionally necessary to replace the entire contents of a file, while retaining the structure of the file itself (as recorded in the header labels). This process is known as modification.

Because all necessary information can be obtained from the file labels, the attach description need only specify that a modify operation is to be performed on a particular file. If a file to be modified does not exist, an error is indicated. The entire contents of the file are replaced by the new data records. The version number in the trailer labels of a modified file is incremented by 1, as described above.

The file to be modified is identified using the -name STR control argument, the -number N control argument, or both in combination. The same rules apply as for reading a file. (See "Reading a File" above.)

If any or all of the structure attribute control arguments are specified, they must match their recorded counterparts; otherwise, an error is indicated.

Generating A File

Recorded in the labels that bracket every file section is a generation number, initially set to 0 when the file is created. The generation number is used to differentiate between different issues (generations) of a file, that all have the same file identifier. The duplicate file identifier rule (see "Creating a File" above) precludes multiple generations of a file from existing simultaneously in the same file set.

The generation number is a higher order of differentiation than the version number, that is more correctly known as the generation version number. While the process of modification or extension does not change the generation number, the process of generation increments the generation number by 1, and resets the version number to 0. The generation number can only be incremented by rewriting the header labels, and it is in this respect that the processes of generation and modification differ.

Producing a new generation of a file is essentially the same as creating a new file in place of the old; however, the file identifier, sequence number, and structure attributes are carried over from the old generation to the new. The attach description need only specify that a generation operation is to be performed on a particular file. If the file to be generated does not exist, an error is indicated. An entirely new generation of the file is created, replacing (and destroying) the previous generation. The generation number is incremented by 1; the version number is reset to 0. When the generation number reaches 9999, the next increment resets it to 0.

The file to be generated is identified by the -name STR control argument, the -number N control argument, or both in combination. The same rules apply as for reading a file. (See "Reading a File" above.)

If any or all of the structure attribute control arguments are specified, they must match those recorded in the labels of the previous generation; otherwise, an error is indicated.

Encoding Mode

The tape_ansi_ I/O module makes provision for three data encoding modes: ASCII, EBCDIC, and binary. Because the DPSR requires that the data in each record be recorded using only ASCII characters, the default data encoding mode is ASCII. File labels are always recorded using the ASCII character set.

When a file is created, the -mode STR can be used to explicitly specify the encoding mode, where STR is the string ascii, ebcdic, or binary. The default is the string ascii. (If -mode STR is not specified, the list_tape_contents command does not supply the specific mode in its report.)

NOTE: THE USE OF ENCODING MODES OTHER THAN ASCII IS A NONSTANDARD FEATURE.

tape_ansi_

If STR is the string ascii, the octal values of the characters to be recorded should be in the range 000 ≤ octal_value ≤ 177; characters in the range 200 to 377 are not invalid, but recording such characters is a nonstandard feature; characters in the range 400 to 777 cause an unrecoverable I/O error. If STR is the string ebcdic, the octal values of the characters to be recorded must be in the range 000 to 177. (See the ascii_to_ebcdic_ subroutine in the MPM Subsystem Writers' Guide for the specific ASCII-to-EBCDIC mapping used by the I/O module.) If STR is the string binary, any octal value can be recorded.

The tape_ansi_ I/O module records the data encoding mode in a portion of the file labels reserved for system-defined use. If the -mode STR control argument is specified when the file is subsequently extended, modified, or generated, the specified mode must match that recorded in the file labels; otherwise, an error is indicated. When the file is subsequently read, the encoding mode is extracted from the file labels, so the -mode STR control argument need not be specified.

File Expiration

Associated with every file is a file expiration date, recorded in the file labels. If a file consists of more than one file section, the same date is recorded in the labels of every section. A file is regarded as "expired" on a day whose date is later than or equal to the expiration date. Only when this condition is satisfied can the file (and by implication, the remainder of the file set) be overwritten. Extension, modification, generation, and the replace mode of creation are all considered to be overwrite operations.

The expiration date is recorded in Julian form; i.e., yyddd, where yy are the last two digits of the year, and ddd is the day of the year expressed as an integer in the range 1 ≤ ddd ≤ 366. A special case of the Julian date form is the value "00000" (always expired).

The expiration date is set only when a file is created or generated. Unless a specific date is provided, the default value "00000" is used. The -expires date control argument is used to specify an expiration date, where date must be of a form acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). If the I/O module is invoked through the iox_$attach_ioname entry point or the iox_$attach_iocb entry point (described in the MPM Subroutines), date must be a contiguous string, with no embedded spaces; if invoked through the io_call command, date may be quoted and contain embedded spaces. Julian form, including "00000", is unacceptable. Because overwriting a file logically truncates the file set at the point of overwriting, the expiration date of a file must be earlier than or equal to the expiration date of the previous file (if any); otherwise, an error is indicated.

If an attempt is made to overwrite an unexpired file, the user is queried for explicit permission. (See "Queries" below). The -force control argument unconditionally grants permission to overwrite a file without querying the user, regardless of "unexpired" status.

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. The volume identifier is a six-character identifier magnetically recorded in the first block of the volume, the VOL1 label. This implementation of the I/O module assumes the volume name and volume identifier to be identical. If this is not the case, the volume identifier must be used in the volume specification field of the attach description.

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.

Occasionally, it is necessary for a user to communicate some additional information to the operator in connection with a mount request. This can be done through the use of the -comment control argument:

vni -comment STR
or
-volume vni -comment STR

where the -comment STR keyword and text specify that a given message is to be displayed on the operator's console whenever volume vni is mounted (a comment can be specified after each volume name supplied). STR can be from 1 to 64 characters. STR can be quoted and contain embedded spaces.

Volume Switching

The DPSR defines four types of file set configurations:

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

tape_ansi_

The tape_ansi_ I/O module maintains a volume sequence list on a per-file-set basis, for the life of a process. A minimal volume sequence list contains only one volume, the first (or only) volume set member. If the file set is a multivolume configuration, the sequence list may contain one or more of the additional volume set members, following the mandatory first volume. If the sequence list contains the entire volume set membership (that may be only one volume), it may then contain one or more volume set candidates. Volume set candidates can become volume set members only as the result of an output operation. When an output operation causes the amount of data in the file set to exceed the capacity of the current volume set membership, the first available volume set candidate becomes a volume set member.

When the first attachment to any file in a file set is made, the volume sequence list for the file set is initialized from the attach description. At detach time, the I/O module empirically determines that one or more volumes are volume set members, by virtue of having used them in the course of processing the attached file. The remaining volumes in the sequence list, if any, are considered to be candidates. In subsequent attachments to any file in the file set, the order of volumes specified in the attach description is compared with the sequence list. For those volumes that the I/O module knows to be volume set members, the orders must match; otherwise, an error is indicated. Those volumes in the sequence list that the I/O module considers to be candidates are replaced by attach description specifications, if the orders differ. If the attach description contains more volumes than the sequence list, the additional volumes are appended to the list. This implementation maintains and validates the volume set membership on a per-process basis, and maintains a list of volume set candidates that is alterable on a per-attach basis.

Once a volume sequence list exists, subsequent attachments to files in the file set do not require repeated specification of any but the first (or only) volume, that is used to identify the file set. If the I/O module detects physical end of tape in the course of an output operation, it prepares to switch to the next volume in the volume set. An attempt is made to obtain the volume name from the sequence list, either from the sublist of members, or the sublist of candidates. If the list of volume set members is exhausted, and the list of candidates is either empty or exhausted, the user is queried for permission to terminate processing. If the reply is negative, the I/O module queries for the volume name of the next volume, which becomes a volume set member and is appended to the volume sequence list. If a volume name is obtained by either method, it is recorded in a system-defined file label field at the end of the current volume, volume switching occurs, and processing of the file continues.

If the I/O module reaches end of file section (but not of file) in the course of an input operation, it first attempts to obtain the next volume name from the volume sequence list. No distinction is made between the member and candidate sublists, because a volume that ends with a file section must be followed by the volume that contains the next section. If the sequence list is exhausted, the file section's labels are examined for a volume name and, if one is found, it is appended to the sequence list. Should the file labels provide no name, the user is queried, as described above. If any of these three methods results in a volume name, volume switching occurs, and processing of the file continues. This method of searching allows a specified switching sequence to override a sequence recorded in the file labels.

If the volume set is demounted at detach time, all volume set candidates are purged from the volume sequence list.

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.

File Set Density

Although the DPSR requires that file sets be recorded at 800 bpi (bits per inch), the I/O module makes provision for three densities: 800, 1600, and 6250 bpi. Every file in a file set must be recorded at the same density; otherwise, an error is indicated.

The -density N control argument is used to explicitly specify the file set density, where N specifies the density at which the file set is (to be) recorded. N can be 800, 1600, 6250 bpi.

NOTE: THE USE OF 1600 OR 6250 BPI IS A NONSTANDARD FEATURE.

The file set density can only be changed in a subsequent attachment if the volume set was demounted by the previous attach.

In the absence of a -density N control argument, the file set density is determined as follows:

```
open for input:  N = density of VOL1 label
open for output, creating new file set:  N = 800 bpi
open for output, old file set: N = density of VOL1 label
```

Opening

The opening modes supported are sequential_input and sequential_output. An I/O switch can be opened and closed any number of times in the course of a single attachment. Such a series of openings may be in either or both modes, in any valid order.

All openings during a single attachment are governed by the same attach description. The following control arguments, all of which pertain to output operations, are ignored when the switch is opened for sequential_input:

```
-create      -generate
-expires     -modify
-extend      -replace
-force
```

tape_ansi_

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 tape_ansi_ 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.

If several attaches and detaches to a file set are made in a process, repeated assignment and unassignment of resources is undesirable. Although the processing time required to assign and unassign a device is small, all available devices can be assigned to other processes in the interval between one detach and the next attach. While volumes are not often "competed" for, mounting and dismounting is both time-consuming and expensive.

The -retain STR control argument is used to specify retention of resources across attachments, where STR specifies the detach-time resource disposition. If STR is the string all, all devices and volumes remain assigned to the process. If STR is the string none, all devices and volumes are unassigned. This is the default retention.

The I/O module provides a further means for specifying or changing the resource disposition subsequent to attachment. If retention of any devices or volumes has been specified at or subsequent to attach time using the retention control operation, the unassign_resource command cannot be used. Instead, use the retain none or retention -none control operation before detaching the I/O module. (See "retention, retain_none, retain_all Operations" under "Control Operations" below.)

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.

If the attach description contains any output control argument (-extend, -modify, -generate, or -create), volumes are mounted with rings; otherwise, they are mounted without rings. When a volume set mounted with rings is opened for sequential_input, hardware file protect is used to inhibit any spurious write operations. A volume set mounted without rings cannot be opened for sequential_output.

However, the following sequence of events is possible. An attach description contains none of the output control arguments, but does contain the -retain all control arguments. The volume set is mounted without rings. After one or more (or no) openings for sequential_input, the I/O switch is detached. The volume set remains mounted because of the -retain all control argument. Subsequently, an attach is made whose description contains an output control argument, that requires that the volume set be mounted with rings. However, as rings can only be inserted in unmounted volumes, the entire volume set must be demounted and then remounted.

This page intentionally left blank.

This situation can be avoided by using the -ring control argument to specify that the volume set be mounted with write rings. If no output control argument is specified in conjunction with -ring, the I/O switch cannot be opened for sequential_output.

When a volume set is mounted with write rings and the I/O switch is opened for sequential_input, the hardware file protect feature is used to safeguard the file set.

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 sequential_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 sequential_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 sequential_output.

query_code = 2 — the first block of the tape is not a valid ANSI VOL1 label. The tape is not formatted as an ANSI volume. This query code can occur only if the I/O stream is opened for sequential_output.

query_code = 3 — the volume identifier recorded in the VOL1 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 sequential_output.

If the I/O stream is opened for sequential_output, the user will be asked whether he wants to initialize or re-initialize the volume. If the I/O stream is opened for sequential_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 sequential_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 sequential_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 must be a volume name six characters or less in length, 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.

Structure Attribute Defaults

When a file is created, the I/O module can supply a default value for any or all of the file structure attributes. The defaults used are as follows:

1. record format the default is f = db
2. block length the default is b = 2048
3. record length
   f = u: undefined
   f = fb | f: r = block length
   f = db | d: r = block length
   f = sb | s: r = 1044480

An injudicious combination of explicit specifications and defaults can result in an invalid attribute set. For example, if the control argument -record 12000 is specified, applying the defaults produces the following:

-format db -block 2048 -record 12000

This attribute set is invalid because, in D format (See "Record Formats" below), the record length must be less than or equal to the block length.

Processing Interchange Files

The DPSR makes provision for recording record format, block length, and record length in specific fields of the HDR2 file label. In addition, the I/O module records the encoding mode in a portion of the HDR2 label reserved for system-defined use. Because the DPSR restricts the encoding mode to ASCII, there is no "standard" label field reserved for recording encoding mode. Therefore, if a foreign interchange file (a file not created by this I/O module) uses an encoding mode other than ASCII, the -mode STR control argument must be used to specify the mode.

File sets are almost always recorded with HDR2 file labels, with the exception of those created by "primitive" systems at implementation levels 1 or 2. (See the DPSR for a description of the facilities supported at different implementation levels.) It is therefore rarely necessary to explicitly specify record format, block length, or record length when interchange files are read, extended, modified, or generated. If, however, a file does lack HDR2 labels, explicit attribute specification is required; defaults apply only to file creation.

ASCII Subset

The DPSR suggests that the characters that comprise certain alphanumeric label fields be limited to a 56-character subset of full ASCII. Furthermore, it is suggested that these fields should not contain embedded blanks, nor should they consist entirely of blanks. In particular, the user need only consider file identifiers and volume names.

The 56-character subset includes:

```
uppercase letters:  ABCDEFGHIJKLMNOPQRSTUVWXYZ
digits:             0123456789
special characters: <space>   " % & ' ( ) * + , - . / : ; < = > ?
```

These characters were chosen from the center four columns of the code table specified in USA Standard Code for Information Interchange, ANSI X3.4-1968, except for position 5/15 (the underscore (_) character) and those positions where there is provision for alternate graphic representation.

The limitation to this subset is intended to provide maximum interchangeability and consistent printing, especially for international interchange.

Overriding Structure Attributes

Normally, the -format f, -block b, and -record r control arguments are not included in the attach description of an I/O switch that is opened for sequential input; the structure attributes are extracted from the file labels. However, the I/O module permits the recorded structure attributes to be overridden by explicitly specified attach description control arguments. Because the apparent structure and characteristics of the file can be drastically altered, great care must be taken to ensure that attribute overrides do not produce unexpected and unwanted results.

If a file has the following recorded attributes:

-format fb -block 800 -record 80

an explicit specification of the -format f and -record 800 control arguments causes each block of ten 80-character records to be treated as a single 800-character record.

If a file has the following recorded attributes:

-format fb -block 800 -record 80

an explicit specification of the -format f, -block 80, and -record 80 control arguments causes the last 720 characters of every block to be discarded. No error is indicated, because every block of the file contains at least one 80-character record.

Record Formats

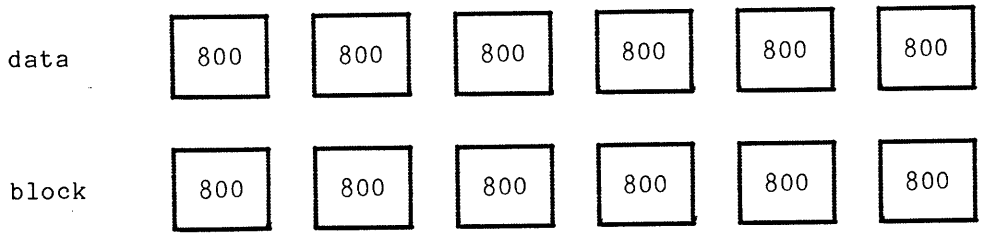
ANSI files are structured in one of three record formats: F, D, or S. In addition, the I/O module provides for a fourth format, U. When a file is created, its record format should be chosen in accordance with the nature of the data to be recorded. For example, data consisting of 80-character card images is most economically recorded in F format, fixed-length records. Data consisting of variable length text lines, such as PL/I source code produced by a text editor, is best recorded in D format, variable-length records. Data of arbitrary length (that could exceed the maximum block size) must be recorded in S format, spanned records, so that a lengthy datum can span several blocks.

F, D, and S format files are either blocked or unblocked, blocked being the normal case. Each block of an unblocked file contains just one record, whereas each block of a blocked file can contain several records. Blocking can provide a significant savings of processing time, because several records are accessed with a single physical tape movement. Furthermore, as blocks are separated by distances of blank tape, blocking reduces the amount of tape needed to contain a file.

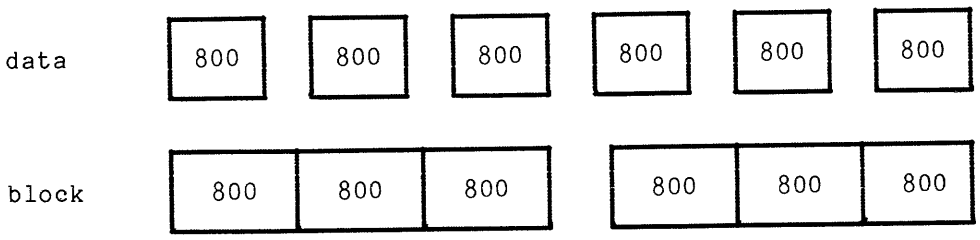
F FORMAT

In F format, records are of fixed (and equal) length, and files have an integral number (n) of records per block. If the file is unblocked, N is equal to 1 and the record length (r) is equal to the block length (b). If the file is blocked, N is greater than 1 and b is equal to (r * N). N is known as the blocking factor.

For example, if r is equal to 800 and b is equal to 800, then the file is unblocked and each block contains just one record.



If r is equal to 800 and b is equal to 2400, then the file is blocked, the blocking factor is 3, and each block contains three records.



The ANSI standard for F format records permits recording a short block only when the last block of a blocked file contains fewer than N records and there are no more records to be written when the file is closed.

There are two special cases in which a datum is padded out to length r. The first case is that of iobl (the iox_$write_record I/O buffer length; i.e., the number of characters to be written) equals 0: a record of r blanks is written. When such a record is subsequently read, it is interpreted as a record of r blanks, and not as a zero-length record. The second case is that of 0 < iobl < r: the record is padded on the right with blanks to length r, and the padded record written. When such a record is read, the original characters plus the padding are returned. The case of iobl is greater than r is in error.

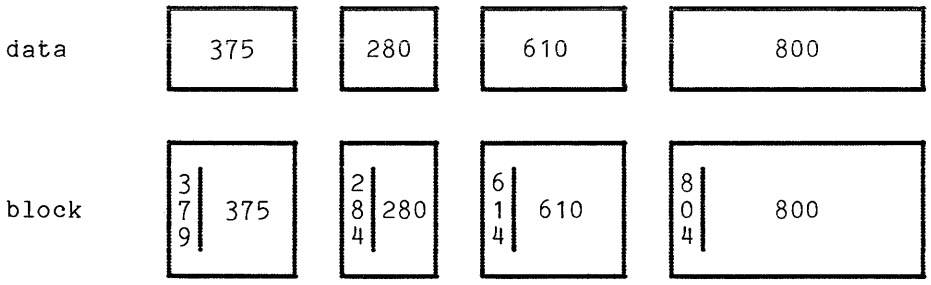
NOTE: THE ANSI STANDARD PROHIBITS RECORDING A FIXED-LENGTH RECORD THAT CONSISTS ENTIRELY OF CIRCUMFLEX (^) CHARACTERS.

D FORMAT

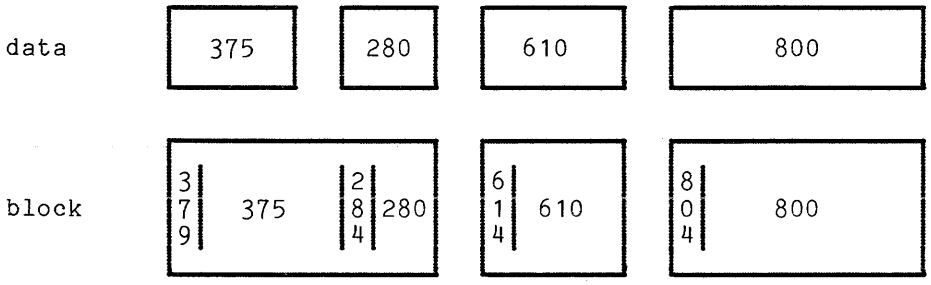
In D format, records and therefore blocks may vary in length. Each record is preceded by a four-character record control word (RCW) that contains the total record length (the length of the data plus the length of the RCW itself).

D format files have an integral number (n) of records per block. If blocked, r is less than or equal to b. For blocked records, the number of records per block varies indirectly with the size of the records.

If r equals b equals 804 and the file is unblocked, records of up to 800 characters can be written, and each block contains one record.



If r equals 804, b is greater than or equal to 804, and the file is blocked, records of up to 800 characters can be written.



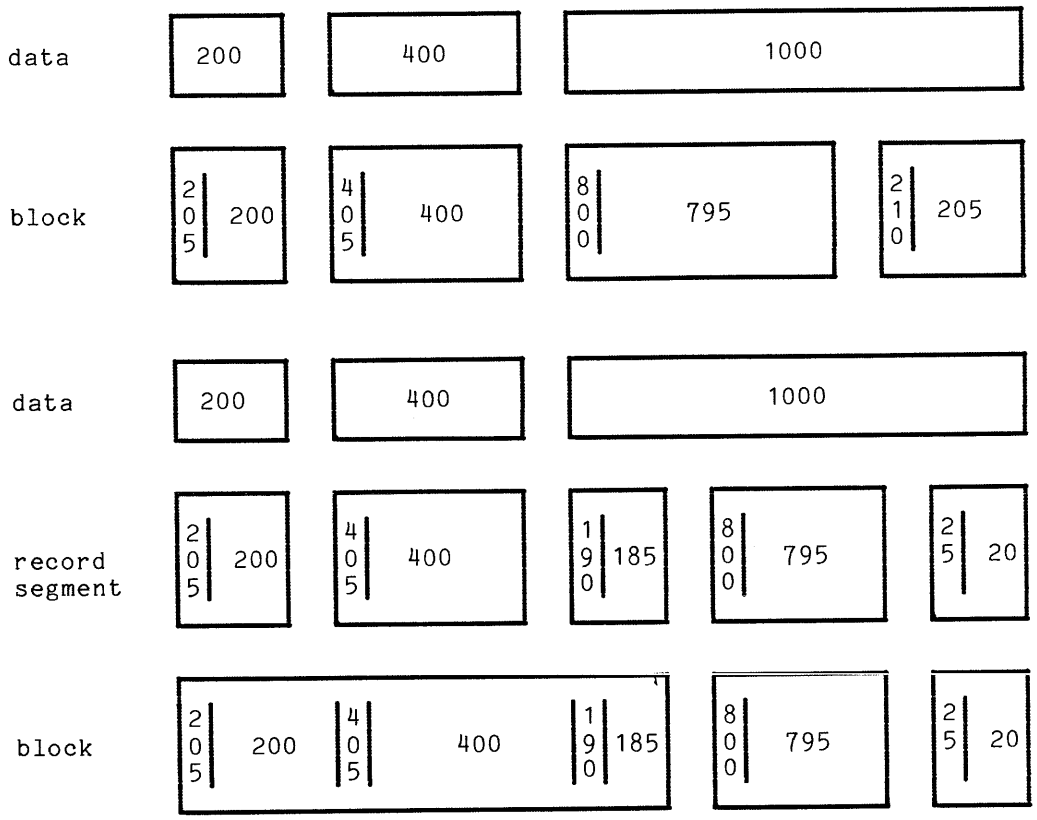
Each block can contain a maximum of 201 zero-length records (a record written as a four-character RCW containing 0004).

S FORMAT

In S format, a single record is formatted as one or more record segments. A record segment contains either a complete record, the initial portion of a record, a medial portion of a record, or the final portion of a record. No two segments of the same record can be contained in the same block, but a block may contain the segments of several different records. The maximum record length is limited only by the maximum size of a storage system segment, currently 1,044,480 characters.

S format files have an integral number of record segments per block. If the file is unblocked, each block contains only one record segment; if blocked, the number of record segments per block is variable. In either case, r and b are independent of one another.

Each record segment begins with a five-character segment control word (SCW). The SCW contains a four-character record segment length, that includes the length of the SCW itself. The SCW also contains a one-character record segment code, that indicates if the segment contains a complete record, or an initial, medial, or final portion. In the examples below, r equals 1000 and b equals 800.



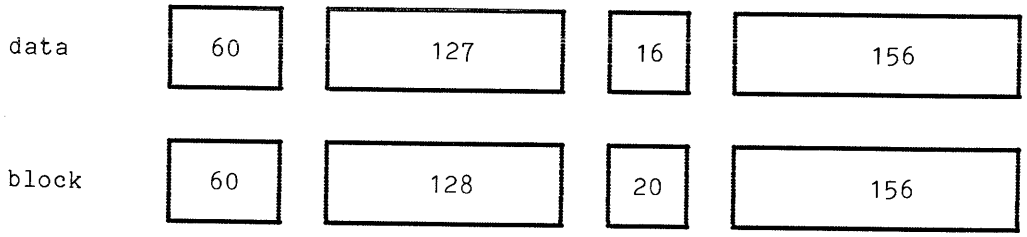
U FORMAT

U format files contain records that do not conform to either F, D, or S format. A U format file is always unblocked. The record length is undefined, and b is greater than or equal to iobl. Blocks may vary in length.

NOTE: THE USE OF U FORMAT IS A NONSTANDARD FEATURE

The ANSI block padding convention permits a block (in any format) to be padded out to any length with circumflex characters (^), according to the requirements of the system that produces the file. These characters are ignored on input. (See "Block Padding" below.) In U format, block padding can lead to an ambiguity; i.e., are trailing circumflexes indeed pad characters, or are they actually valid data within the nonpadded portion of the block. The DPSR suggests that a U format block be treated as a single record. In conformance with this suggestion, the I/O module considers trailing circumflexes to be valid data.

The special case of writing a record where iobl is less than 20 characters produces a block padded to length 20 with circumflex characters.



Record Format Comparison

At first glance, it might appear as if S format were the format of choice, simply because it has the fewest restrictions and the greatest flexibility. Although the latter is certainly true, the former is by no means a valid inference. Increased flexibility is almost invariably accompanied by decreased processing efficiency.

F format requires the least processing time, and should be used if the records are fixed-length. If F format is used with nonfixed-length records the record padding rules apply, so the user must ensure that recorded data is not irretrievably (and perhaps undetectably) modified.

D format, with explicit inclusion of record length in the RCW, is perhaps the "safest" format to use: there are no special padding cases, and the RCW provides an additional validity check. The D format processing overhead is small.

S format permits almost any datum to be recorded, irrespective of length, and further has the "safety" advantage of D format because each segment includes an SCW. While S format records provide maximum flexibility, their use entails considerably more processing time than the use of F or D format.

Block Padding

The DPSR makes provision for extending the recorded length of a block beyond the end of the last (or only) record whenever such padding is deemed necessary or advisable. Padding characters are not considered when computing an RCW or SCW length. Because the Multics system is implemented on a word-oriented computer, the number of characters in a block must be evenly divisible by four. The I/O module automatically pads every block to the correct length, using from 1 to 3 circumflex characters. In addition, the DPSR does not permit recording a block of fewer than 18 characters. To conform with this requirement, the I/O module pads any block containing fewer than 20 characters out to length 20.

As long as F, D, or S format is used, the presence or absence of block padding characters in a particular block is user-transparent. If U format is used, it is the responsibility of the user to detect and ignore any pad characters that may be generated.

tape_ansi_

Volume Initialization

The DPSR requires that all volumes be initialized with a VOL1 label and dummy file before they are used for output. The I/O module provides a semiautomatic volume initialization mechanism that performs this operation as an integral part of the output function. The rules that govern permission to initialize a volume are complex, and permission to initialize under most circumstances is specifically denied (by the DPSR) to the application program. The I/O module's mechanism strikes a balance between outright denial and absolute ease. (See "Queries" above.)

It should be noted that a newly initialized volume contains a dummy file. Thus, if a file is created on a newly initialized volume without an explicit specification of the -number 1 control argument, the file is appended to the file set, resulting in a file sequence number of 2, and not 1 as might be expected.

Buffer Offset (Block Prefix)

The DPSR provides for each block of a file being prefixed by from 1 to 99 characters of prefix information, known as the buffer offset. The buffer offset length is recorded in the HDR2 label. If an input file has block prefixes, and the block length is explicitly specified, it must be incremented by the buffer offset length. This calculation should made after the block length has been determined using the normal block-record relationship rules.

The I/O module ignores (skips) buffer offsets on input, and does not provide for writing buffer offsets on output, except when extending or modifying an interchange file with a nonzero buffer offset. In this case, each block written is prefixed with an appropriate number of blanks.

Conformance To Standard

The I/O module conforms to the ANSI standard for level 4 implementations with the following five exceptions:

1. Volume Initialization -- The I/O module has a permission-granting mechanism that can be controlled by the application program.

2. Volume and File Accessibility -- On input, the I/O module always grants permission to access. On output, the access control fields in the VOL1 and HDR1 labels are always recorded as blank (" ").

3. Overwriting Unexpired Files -- The I/O module has a permission-granting mechanism that can be controlled by the application program.

4. User Label Processing -- The I/O module ignores user labels on input, and does not provide for writing user labels on output.

5. Buffer Offset Processing -- The I/O module ignores buffer offsets on input, and does not provide for writing buffer offsets on output (except as stated above).

Label Processing

VOL1

The label is processed on input and output. The owner-identifier field, character positions (CP) 38 to 51, holds a three-character volume authentication code.

UVLa

These labels are not written on output, and ignored on input.

HDR1/EOF1/EOV1

The labels are processed on input and output. The system-code field, CP 61 to 73, is recorded as "MULTICS ANSI ".

HDR2/EOF2/EOV2

The labels are processed on input and output. The reserved-for-system-use field, CP 16 to 50, is recorded as follows:

```
CP 16 to 47 - full 32-character volume name of next volume (EOV2 only)
CP 48       -   blocking attribute (all)
                "0" = unblocked; "1" = blocked
CP 49       -   data encoding mode (all)
                "1" = ASCII, 9 mode
                "2" = EBCDIC, 9 mode
                "3" = binary
```

HDR3/EOF3/EOV3 - HDR9/EOF9/EOV9

These labels are not written on output and are ignored on input.

UHLa/UTLa

These labels are not written on output and are ignored on input.

tape_ansi_

Error Processing

If an error occurs while reading, the I/O module makes 25 attempts to backspace and reread. If an error occurs while writing, the I/O module makes 10 attempts to backspace, erase, and rewrite. Should an unrecoverable error occur while reading or writing the, I/O module "locks" the file so that no further I/O is possible. (See reset_error_lock OPERATION, below.) If an unrecoverable error occurs while writing file labels or tapemarks, the user is queried about preserving the defective file versus file set consistency. (See "Queries" above.) 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

Close Operation

The I/O switch must be open.

Control Operation

The I/O module supports eleven control operations.

hardware_status	close_rewind
status	retention
	retain_none
	retain_all
volume_status	reset_error_lock
file_status	volume_density
feov	

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

hardware_status OPERATION

This operation returns the 72-bit IOM status string generated by the last tape I/O operation. The I/O switch must be open. The substr argument (IOM_bits, 3, 10) contains the major and minor status codes generated by the tape subsystem itself. (See MTS500 Magnetic Tape Subsystem, Order No. DB28, for an explanation of major and minor status.) The variable to which info_ptr points is declared as follows:

```
declare IOM_bits bit(72) aligned;
```

status OPERATION

This operation returns a structure that contains an array of status codes, providing an interpretation of the IOM status string generated by the last tape I/O operation. These codes may be used in calls to the com_err_ subroutine, or may be converted to printable strings by calling the convert_status_code_ subroutine. (See the description of the com_err_ subroutine in the MPM Subroutines and the description of the convert_status_code_ subroutine in the MPM Subsystem Writers' Guide.) The I/O switch must be open. The structure to which info_ptr points, device_status.incl.pl1, is declared as follows:

```
dcl  dstat_ptr        pointer;
dcl 1 device_status   based (dstat_ptr),
    2 IOM_bits        bit(72) aligned,     /* IOM status */
    2 n_minor         fixed bin,           /* number of minor codes */
    2 major           fixed bin(35),       /* major status code */
    2 minor           (10) fixed bin(35); /* minor status codes */
```

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. If the switch was closed by the I/O module as the result of an error while writing file header labels, trailer labels, or tapemarks, the current volume is the last (or only) volume in the volume set. The structure to which info_ptr points, tape_volume_status.incl.pl1, is declared as follows:

```
dcl  tvstat_ptr           pointer;
dcl 1 tape_volume_status  based (tvstat_ptr),
    2 volume_name         char(6),       /* volume name */
    2 volume_id           char(6),       /* from VOL1 label */
    2 volume_seq          fixed bin,     /* order in volume set */
    2 tape_drive          char(8),       /* tape drive name */
                                         /* "" if not mounted */
    2 read_errors         fixed bin,     /* read error count */
    2 write_errors        fixed bin;     /* write error count */
```

In the current implementation of the I/O module, read_errors and write_errors are always zero. Eventually, the resource control package (RCP) supplies these values.

file_status OPERATION

This operation returns a structure that contains the current status of the file specified in the attach description. If the I/O switch has never been opened, no information can be returned; this situation is indicated by tape_file_status.state = 0. If the switch was opened, but is now closed, the current status of the file is its status just prior to closing. If the switch was closed by the I/O module as the result of an error while writing file header labels, trailer labels, or tapemarks, the entire file may have been deleted. In this case, the structure contains the current status of the previous file in the file set, if any. The structure to which info_ptr points, tape_file_status.incl.pl1, is declared as follows:

```
dcl  tfstat_ptr           pointer;
dcl 1 tape_file_status    based (tfstat_ptr),
    2 state               fixed bin,     /* 0 - no information */
                                         /* 1 - not open */
                                         /* 2 - open, no events */
                                         /* 3 - open, event lock */
    2 event_code          fixed bin(35), /* error_table_ code if
                                            state = 3 */
    2 file_id             char(17),      /* file identifier */
    2 file_seq            fixed bin,     /* order in file set */
    2 cur_section         fixed bin,     /* current or last
                                            section processed */
    2 cur_volume          char(6),       /* volume name of volume
                                            on which cur_section
                                            resides */
    2 generation          fixed bin,     /* generation number */
    2 version             fixed bin,     /* version of generation */
    2 creation            char(5),       /* Julian creation date */
    2 expiration          char(5),       /* Julian expiration date */
    2 format_code         fixed bin,     /* 1 - U format */
                                         /* 2 - F format */
                                         /* 3 - D format */
                                         /* 4 - S format */
    2 blklen              fixed bin,     /* block length */
    2 reclen              fixed bin(21), /* record length */
    2 blocked             bit(1),        /* "0"b - no | "1"b - yes */
    2 mode                fixed bin,     /* 1 - ASCII */
                                         /* 2 - EBCDIC */
                                         /* 3 - binary */
    2 cur_blkcnt          fixed bin(35); /* current block count */
```

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

feov OPERATION

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

close_rewind OPERATION

This operation specifies that the current volume is to be rewound when the I/O switch is next closed. The info_ptr should be a null pointer. The switch need not be open when the operation is issued. The operation effects only one close; subsequent closings require additional control calls.

retention, retain_none, retain_all OPERATIONS

These operations cause the tape resources currently in use, i.e., tape drives(s) and tape volume(s), to be unassigned or retained at detach time according to the specified retention argument or operation. The info_ptr points to a fixed binary number with value as defined below:

1 retention -none or retain_none
causes none of the tape resources currently in use to remain assigned at detach time.

2 retention -volume
causes the tape volume(s) currently in use to remain assigned at detach time.

3 retention -device
causes the tape drives(s) currently in use to remain assigned at detach time.

4 retention -all or retain_all
causes all of the devices and volumes currently in use to remain assigned at detach time.

reset_error_lock OPERATION

This operation unlocks the files so that further I/O is possible subsequent to a parity-type I/O error while reading. Such an error is indicated by a previous iox $read_record or iox $position call having returned the status code error_table_$tape_error. In this case, the value of tape_file_status.event_lock is error_table_$tape_error. (See file_status OPERATION, above.) The I/O switch must be open for sequential_input. The info_ptr should be a null pointer.

NOTE: IF RECORDS ARE BLOCKED AND/OR SPANNED, THE VALIDITY OF ANY RECORDS READ SUBSEQUENT TO A PARITY-TYPE I/O ERROR IS NOT GUARANTEED. (The parity error is reported for the first read of a logical record in the block. The actual location of the error in the block is unknown.)

volume_density OPERATION

This operation returns the encoded density of the volume set. The I/O switch need not be open. The variable to which info_ptr points is declared as follows:

```
declare volume_density fixed bin;
```

The values returned and their meanings are listed below:

value	meaning
-1	none specified yet
2	800
3	1600
4	6250

Detach Operation

The I/O switch must be closed. 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.

Modes Operation

This I/O module does not support the modes operation.

Position Operation

The I/O switch must be open for sequential_input. The I/O module does not support skipping backwards. In the course of a position operation, events or errors may occur that invoke the query mechanism. (See "Queries" above.) An unrecoverable error locks the file, and a severe error causes the I/O module to close the I/O switch.

Read Length Operation

The I/O switch must be open for sequential_input. In the course of a read_length operation, events or errors may occur that invoke the query mechanism. (See "Queries" above.) An unrecoverable error locks the file, and a severe error causes the I/O module to close the I/O switch.

Read Record Operation

The I/O switch must be open for sequential_input.

Write Record Operation

The I/O switch must be open for sequential_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 and summarized below.

operation	abbreviation	control_arg
status	st	-all
hardware_status	hst	
reset_error_lock	rel	
file_status	fst	
volume_status	vst	
retention	ret	-none, -volume, -device, -all
retain_all	reta	
retain_none	retn	
close_rewind	crw	
feov	feov	

3. control_arg
   is an operation control argument valid only for the retention and the status operations. A control argument is required for the retention operation; possible control arguments are described below.

   -none
   causes none of the tape resources currently in use to remain assigned at detach time.

   -volume
   causes the tape volume(s) currently in use to remain assigned at detach time.

   -device
   causes the tape drives(s) currently in use to remain assigned at detach time.

   -all
   causes all of the devices and volumes currently in use to remain assigned at detach time.

   The -all control argument is optional for the status operation. This control argument prints all available tape status information such as the device status, the volume status, the file status, and the hardware status. The -all control argument is only for use with the status operation through the io_call command. It is not defined for use in the status operation with iox_$control directly.

tape_ansi_

Examples

In the following examples, it must be emphasized that an attach description describes a potential operation, and in and of itself does nothing to the file. Depending upon the sequence of openings in various modes, one attach description can perform diverse functions.

```
tape_ansi_ 042381 -nm ARD21 -cr -fmt sb -ret all
```

A file named ARD21 is to be appended to the file set whose first volume is 042381. If a file named ARD21 already exists in the file set, openings for sequential_input access that file, and openings for sequential_output create new files replacing the old. If no file named ARD21 already exists in the file set, openings for sequential_input prior to the first opening for sequential_output fail. The first opening for sequential_output creates the file by appending it to the end of the file set. Subsequent openings for sequential_input access the newly created file, and subsequent openings for sequential_output replace it. Spanned records are specified; the block length defaults to 2048, the record length to 1044480, and the encoding mode to ASCII. The density defaults to 800 bpi, and the maximum number of devices defaults to 1. The volume set and devices are retained after detachment.

```
tape_ansi_ 042381 -nm fargo.pl1 -nb 2 -cr -force -fmt fb -bk 800 -rec 80
```

A file named fargo.pl1 is created at position 2 in the file set. If a file named fargo.pl1 already exists at position 2, openings for sequential_input prior to the first opening for sequential_output access that file. The first opening for sequential_output creates a new file, and subsequent openings for sequential_input access the new file. If no file named fargo.pl1 exists at position 2, openings for sequential_input prior to the first opening for sequential_output fail. If a file exists at position 2, it is replaced irrespective of its expiration date.

```
tape_ansi_ 042381 -nm zbx -rpl zbx -cr -md binary -bk 6000 -exp 2weeks
```

A file named zbx is to be created, replacing a file of the same name. Openings for sequential_input prior to the first opening for sequential_output access the old file. Each opening for sequential_output creates a new file, and each subsequent opening for sequential_input accesses the most recently created file. The specified encoding mode is binary. The record format defaults to D, blocked, and the record length defaults to 6000 because the block length is specified as 6000. The file is protected from overwriting for a period of two weeks, so each opening for sequential_output subsequent to the initial opening for sequential_output causes the user to be queried for permission to overwrite.

```
tape_ansi_ 042381 -nb 14 -gen -dv 3 -expires 12/31/77
```

A new generation of the file at position 14 in the file set is to be created, replacing the old generation. If the old generation is not expired, the user is queried for permission to overwrite. Each opening for sequential_input accesses the current generation. Each opening for sequential_output creates a new generation. The new generation has an expiration date of December 31, 1977. The maximum number of devices that can be used is three.

```
tape_ansi_ 042381 042382 042383 -nm THESIS -rg
```

A file named THESIS is to be read. The I/O switch can only be open for sequential_input. The volume set consists of at least three volumes, and they are mounted with write rings. Only one device can be used.

```
tape_ansi_ 042381 -nm FF -nb 3 -ext -dv 4 -ret all
```

A file named FF at position 3 in the file set is to be extended. Each opening for sequential_input accesses the current version. Each opening for sequential_output produces a new version. A maximum of four devices can be used, and resources are retained after detachment.

```
tape_ansi_ 042381 -vol -COS -com in_slot_000034 -nb 6 -mod -fc
```

The file at position 6 in the file set is to be modified, irrespective of its expiration date. Each opening for sequential_input accesses the current version. Each opening for sequential_output produces a new version. The second volume of the volume set has volume identifier -COS, and can be found in slot 000034.

Attach Control Arguments

The following is a complete list of all valid attach control arguments in both long and short forms:

-block b	-bk b	18 ≤ b ≤ 99996
-clear	-cl	
-create	-cr	
-density N	-den N	N = 800 ¦ 1600 ¦ 6250
-device N	-dv N	1 ≤ N ≤ 63
-expires date	-exp date	valid date
-extend	-ext	
-force	-fc	
-format f	-fmt f	f = fb ¦ f ¦ db ¦ d ¦ sb ¦ s ¦ u
-generate	-gen	
-mode STR	-md STR	STR = ascii ¦ ebcdic ¦ binary
-modify	-mod	
-name STR	-nm STR	STR ≤ 17 characters
-number N	-nb N	1 ≤ N ≤ 9999
-record r	-rec r	1 ≤ r ≤ 1044480
-replace	-rpl	STR ≤ 17 characters
-retain STR	-ret STR	STR = all ¦ none
-ring	-rg	

The following is a list of positional keywords:

-comment STR	-com STR	STR ≤ 64 characters
-volume vni	-vol vni	vni ≤ 6 characters

Name: tape_ibm_

The tape_ibm_ I/O module implements the processing of magnetic tape files in accordance with the standards established by the following IBM publications: OS Data Management Services Guide, Release 21.7, GC26-3746-2; IBM System 360 Disk Operating System Data Management Concepts, GC24-3427-8; and OS Tape Labels, Release 21, GC28-6680-4. These documents are collectively referred to below as the Standard.

Entries in the 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

record
: related information treated as a unit of information.

block
: a collection of characters written or read 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 records pertaining to a single subject. A file may be recorded on all or part of a volume, or on more than one volume.

volume
: 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.

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

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

Attach Description

The attach description has the following form:

```
tape_ibm_ vn1 vn2 ... vnN {-control_args}
```

tape_ibm_

where:

1. vn*i*

   is a volume specification. A maximum of 64 volumes may be specified. In the simplest (and typical) case, a volume specification is a volume name that must be six characters or less in length. If a volume name is less than six characters and entirely numeric, it is padded on the left with 0's. If a volume name is less than six characters and not entirely numeric, it is padded on the right with blanks. Occasionally, keywords must be used with the volume name. For a discussion of volume name and keywords see "Volume Specification" below.

   vn*1* vn*2* ... vnN

   comprise what is known as the volume sequence list. The volume sequence list may be divided into two parts. The first part, vn*1* ... vn*i*, 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, vn*i+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

   may be one or more attach control arguments. A control argument may appear only once.

   -block *b*, -bk *b*

   specifies the block length in characters, where the value of *b* is dependent upon the value of *r* specified in the -record control argument. (See "Creating A File" below.)

   -clear, -cl

   specifies that internal information on a file-set which the I/O module retains from previous attachments is to be deleted. This control argument can be used when it is desired to change attributes of a file-set which are maintained across attachments for a given process, e.g. density or label standard. For the initial attachment to a file-set in a given process, this control argument has no effect.

   -create, -cr

   specifies that a new file is to be created. (See "Creating A File" below.)

   -density N, -den N

   specifies the density at which the file set is recorded, where N can be 800, 1600, or 6250 bits per inch. (See "File Set Density" below.)

   -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 $1 \leq N \leq 63$. (See "Multiple Devices" below.)

   -dos

   specifies that a file was produced by, or is destined for, a DOS installation. (See "DOS Files" below.)

-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. (See "File Expiration" below.)

-extend, -ext
specifies extension of an existing file. (See "Extending a File" below.)

-force, -fc
specifies that the expiration date of the file being overwritten is to be ignored. (See "File Expiration" below.)

-format f, -fmt f
specifies the record format, where f is a format code. (See "Creating A File" below for a list of format codes.)

-mode STR, -md STR
specifies the encoding mode used to record the file data, where STR is the string ebcdic, ascii, or binary; the default is ebcdic. (See "Encoding Mode" below.)

-modify, -mod
specifies modification of an existing file. (See "Modifying a File" below.)

-name STR, -nm STR
specifies the file identifier of the file, where STR is from 1 to 17 characters. (See "Creating A File" below.)

-no_labels, -nlb
specifies that unlabeled tapes are to be processed. (See "Unlabeled Tapes" below.)

-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 $1 \leq N \leq 9999$. (See "Creating A File" below.)

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

-replace STR, -rpl STR
specifies the file identifier of the file to be replaced, where STR must be from 1 to 17 characters. If no file with file identifier STR exists, an error is indicated. (See "Creating A File" below.)

-retain STR, -ret STR
specifies retention of resources across attachments, where STR specifies the detach-time resource disposition. (See "Resource Disposition" below.)

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

tape_ibm_

-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.)

The following sections define each control argument in the contexts in which it can be used. For a complete list of the attach control arguments see "Attach Control Arguments" below.

File Identifiers

Associated with every file is a name (file identifier) and a number (file sequence number). The file identifier must be 17 characters or less. When creating a file, the file identifier must be composed of one or more components of one to eight characters, with adjacent components separated by a period. The first character of each component must be an uppercase letter or national character (@, #, or $) and the remaining characters must be uppercase letters, national characters or the digits 0 to 9. If a file identifier (of an existing file) does not meet the naming conventions established for files created on the Multics system, the file must be referenced using the -number control argument and a file sequence number.

Creating A File

When a file is created, an entirely new entity is added to the file set. There are two modes of creation: append and replace. In append mode, the new file is added to the file set immediately following the last (or only) file in the set. The process of appending does not alter the previous contents of the file set. In replace mode, the new file is added by replacing (overwriting) a particular previously existing file. The replacement process logically truncates the file set at the point of replacement, destroying all files (if any) that follow consecutively from that point.

The -create and -name control arguments are required to create a file, where STR is the file identifier. If no file having file identifier STR exists in the file set, the new file is appended to the file set; otherwise, the new file replaces the old file of the same name.

If the user wishes to explicitly specify creation by replacement, the particular file to be replaced must be identified. Either a file identifier or a file sequence number is sufficient to uniquely identify a particular file in the file set. The -number and -replace control arguments either separately or in conjunction, are used to specify the file to be replaced. If used together, they must both identify the same file; otherwise, an error is indicated.

When the -number control argument is specified, if N is less than or equal to the sequence number of the last file in the file set, the created file replaces the file having sequence number N. If N is one greater than the sequence number of the last file in the file set, the created file is appended to the file set. If N is any other value, an error is indicated. When creating the first file of an entirely new file set, the -number control argument must be explicitly specified. (See "Volume Initialization" below.)

The -format, -record and -block control arguments are used to specify the internal structure of the file to be created. They are collectively known as structure attribute control arguments. When the -format control argument is used, f must be one of the following format codes, chosen according to the nature of the data to be recorded. (For a detailed description of the various record formats, see "Record Formats" below.)

fb for fixed-length records. Used when every record has the same length, not in excess of 32760 characters.

vb for variable-length records. Used when records are of varying lengths, the longest not in excess of 32752 characters.

vbs for spanned records. Used when the record length is fixed and in excess of 32760 characters, or variable and in excess of 32752 characters. In either case, the record length cannot exceed 1,044,480 characters. (See "DOS Files" below.)

f for fixed-length records, unblocked.

v for variable-length records, unblocked.

vs for spanned records, unblocked. (See "DOS Files" below.)

NOTE: Because of padding requirements records recorded using vs format may be irreversibly modified. (See "Padding" below.)

Unblocked means that each block contains only one record (f, v) or record segment (vs). Because of their relative inefficiency, the use of unblocked formats in general is discouraged. Blocked means that each block contains as many records (fb, vb) or record segments (vbs) as possible. The actual number of records/block is either fixed (fb), depending upon the block length and record length, or variable (vb, vbs), depending upon the block length, record length, and actual records.

u for undefined records. U format records are undefined in format. Each block is treated as a single record, and a block may contain a maximum of 32760 characters.

When the -record control argument is used, the value of r is dependent upon the choice of record format. In the following list, amrl is the actual or maximum record length.

f = fb ¦ f:	r = amrl
f = vb ¦ v:	amrl + 4 ≤ r ≤ 32756
f = vbs ¦ vs:	amrl ≤ r ≤ 1044480
f = u:	r is undefined (the -record control argument should not be used.)

tape_ibm_

When the -block control argument is used, the value of b is dependent upon the value of r. When the block length is not constrained to a particular value, the largest possible block length should be used.

f = fb:	b must satisfy mod (b,r) = 0
f = f:	b = r
f = vb:	b ≥ r + 4
f = v:	b = r + 4
f = vbs | vs:	20 ≤ b ≤ 32760
f = u:	amrl ≤ b ≤ 32760

In every case, b must be an integer in the range 20 ≤ b ≤ 8192, and, when the I/O switch is opened for sequential_output, must satisfy mod (b,4) = 0.

Since the structure attribute control arguments are interdependent, care must be taken to ensure that specified values are consistent.

Padding

Since the Multics system is implemented on word-oriented hardware, records recorded in any format are subject to block and/or record padding. On output, the hardware requires that the number of characters in a block be evenly divisible by 4; i.e., only words can be written. The I/O module therefore requires that mod (b,4) = 0, and pads a record, if necessary, to meet this requirement. (Warning: this padding may cause IBM-system rejection of a block if block length is not a multiple of the record length.) The following rules govern padding on output:

f = fb: if iobl (the I/O buffer length in an iox_$write_record call; i.e., the number of characters to be written) is less than r, the record is padded on the right with blanks to length r. The last (or only) record of the file may be padded on the right with N blanks, where 0 ≤ N ≤ 19 is sufficient to satisfy b ≥ 20, and mod (b,4) = 0.

f = f: if iobl is less than r, the record is padded on the right with blanks to length r. Because the specified value of b must satisfy b ≥ 20, mod (b,4) = 0, and r = b, there are no other padding possibilities.

f = vb: the last (or only) record in every block is padded on the right with N blanks, where 0 ≤ N < 12 is sufficient to satisfy b ≥ 20, and mod (b,4) = 0. Because the number of records in a block is variable, it is difficult to determine which records of a file are padded, if any.

f = v: every record is padded on the right with N blanks, where 0 ≤ N ≤ 12 is sufficient to satisfy b ≥ 20, and mod (b,4) = 0.

f = vbs: the last (or only) record of the file is padded on the right with N blanks, where 0 ≤ N ≤ 12 is sufficient to satisfy b ≥ 20, and mod (b,4) = 0.

f = vs: every record or record segment is padded on the right with N blanks, where $0 \leq N \leq 12$ is sufficient to satisfy $b \geq 20$, and mod (b,4) = 0.

NOTE: This requirement can result in an indeterminate number of blanks being inserted into a record at one or more indeterminate positions.

f = u: every record is padded on the right with N blanks, where $0 \leq N \leq 12$ is sufficient to satisfy $b \geq 20$, and mod (b,4) = 0.

Reading A File

The attach description needed to read a file is less complex than the description used to create it. When a file is initially created by the I/O module, the structure attributes specified in the attach description are recorded in the file's header and trailer labels. These labels, that precede and follow each file section, also contain the file name, sequence number, block count, etc. Files created by OS installations also record the structure attributes in the file labels. (See "DOS Files" below.) When a file is subsequently read, all this information is extracted from the labels. Therefore, the attach description need only identify the file to be read; no other control arguments are necessary.

The file can be identified using the -name control argument, the -number control argument, or both in combination. If the -name control argument is used, a file with the specified file identifier must exist in the file set; otherwise, an error is indicated. If the -number control argument is used, a file with the specified file sequence number must exist in the file set; otherwise, an error is indicated. If the -name and -number control arguments are used together, they must both refer to the same file; otherwise, an error is indicated.

DOS Files

Files created by DOS installations differ from OS files in one major respect -- DOS does not record HDR2 labels, which contain the structure attributes. It is therefore necessary to specify all of the structure attributes whenever a file created by a DOS installation is to be processed.

It is further necessary to distinguish between OS and DOS files recorded in VBS or VS format. The segment descriptor word (SDW) of a zero-length DOS spanned record has a high-order null record segment bit set, while a zero-length OS spanned record does not. (See "V(B)S Format" below, for an explanation of the SDW.)

The -dos control argument must be used when writing a VBS or VS file destined for a DOS installation, or when reading a VBS or VS file written by a DOS installation. In the interest of clarity, however, it is recommended that the control argument always be specified when DOS files are processed, regardless of record format.

tape_ibm_

Output Operations On Existing Files

There are two output operations that can be performed on an already existing file: extension and modification. As their functions are significantly different, they are described separately below. They do, however, share a common characteristic. Like the replace mode of creation, an output operation on an existing file logically truncates the file set at the point of operation, destroying all files (if any) that follow consecutively from that point. Because the block length is constrained to mod(b,4) = 0 for output operations, a file whose block length does not satisfy this criterion cannot be extended or modified.

Extending A File

It is often necessary to add records to a file without in any way altering the previous contents of the file. This process is known as extension.

Because all the information regarding structure, length, etc., can be obtained from the file labels, the attach description need only specify that an extend operation is to be performed on a particular file. (See "DOS Files" above.) If the file to be extended does not exist, an error is indicated. New data records are appended at the end of the file; the previous contents of the file remain unchanged.

The file to be extended is identified using the -name control argument, the -number control argument, or both in combination. The same rules apply as for reading a file. (See "Reading a File" above.)

The user may specify any or all of the structure attribute control arguments when extending a file. The specified control arguments are compared with their recorded counterparts; if a discrepancy is found, an error is indicated.

Modifying A File

It is occasionally necessary to replace the entire contents of a file, while retaining the structure of the file itself. This process is known as modification.

Because all necessary information can be obtained from the file labels, the attach description need only specify that a modify operation is to be performed on a particular file. (See "DOS Files" above.) If a file to be modified does not exist, an error is indicated. The entire contents of the file are replaced by the new data records.

The file to be modified is identified using the -name control argument, the -number control argument, or both in combination. The same rules apply as for reading a file. (See "Reading a File" above.)

If any or all of the structure attribute control arguments are specified, they must match their recorded counterparts; otherwise, an error is indicated.

Encoding Mode

The I/O module makes provision for three data encoding modes: EBCDIC, binary, and ASCII. The default data encoding mode is EBCDIC. File labels are always recorded using the EBCDIC character set.

When a file is created, the -mode control argument can be used to explicitly specify the encoding mode (if not used, the list_tape_contents command does not supply the specific mode in its report).

If STR is the string ascii, the octal values of the characters to be recorded must be in the range 000 ≤ octal_value ≤ 377; otherwise, an unrecoverable I/O error occurs. If STR is the string ebcdic, the octal values of the characters to be recorded must be in the range 000 ≤ octal_value ≤ 177. (See the ascii_to_ebcdic_ subroutine in the MPM Subsystem Writers' Guide for the specific ASCII to EBCDIC mapping used by the I/O module.) If STR is the string binary, any 9-bit byte value can be recorded. However, data written on IBM equipment with binary mode may not be compatible with Multics, or vice versa.

Because the data encoding mode is not recorded in the file labels, the -mode ascii and the -mode binary control arguments must always be specified when subsequently processing an ASCII or binary file, respectively.

File Expiration

Associated with every file is a file expiration date, recorded in the file labels. If a file consists of more than one file section, the same date is recorded in the labels of every section. A file is regarded as "expired" on a day whose date is later than or equal to the expiration date. Only when this condition is satisfied can the file (and by implication, the remainder of the file set) be overwritten. Extension, modification, and the replace mode of creation are all considered to be overwrite operations.

The expiration date is recorded in Julian form; i.e., yyddd, where yy are the last two digits of the year, and ddd is the day of the year expressed as an integer in the range 1 ≤ ddd ≤ 366. A special case of the Julian date form is the value "00000", which means always expired.

The expiration date is set only when a file is created. Unless a specific date is provided, the default value "00000" is used. The -expires control argument is used to specify an expiration date where date must be of a form acceptable to the convert_date_to_binary_ subroutine (described in the MPM Subroutines). If the I/O module is invoked through the iox_$attach_ioname entry point or the iox_$attach_iocb entry point, date must be a contiguous string, with no embedded spaces; if invoked through the io_call command, date may be quoted and contain embedded spaces. Julian form, including "00000", is unacceptable. Because overwriting a file logically truncates the file set at the point of overwriting, the expiration date of a file must be earlier than or equal to the expiration date of the previous file (if any); otherwise, an error is indicated.

If an attempt is made to overwrite an unexpired file, the user is queried for explicit permission. (See "Queries" below). The -force control argument unconditionally grants permission to overwrite a file without querying the user, regardless of "unexpired" status.

Volume Specification

The volume name (also called the slot identifier) is an identifier physically written on, or affixed to, the reel or container of the volume. The volume identifier is a six-character identifier magnetically recorded in the first block of the volume, the VOL1 label. This implementation of the I/O module assumes the volume name and volume identifier to be identical. If this is not the case, the volume identifier must be used in the volume specification field of the attach description.

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 -volume 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.

Occasionally, it is necessary for a user to communicate some additional information to the operator in connection with a mount request. This can be done through the use of the -comment control argument:

vni -comment STR
or
-volume vni -comment STR

where the -comment STR keyword and text specify that a given message is to be displayed on the operator's console whenever volume vni is mounted (a comment can be specified after each volume name supplied). STR can be from 1 to 64 characters. STR can be quoted and contain embedded spaces.

Volume Switching

The Standard defines four types of file set configurations:

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

tape_ibm_

The I/O module maintains a volume sequence list on a per-file-set basis, for the life of a process. A minimal volume sequence list contains only one volume, the first (or only) volume set member. If the file set is a multivolume configuration, the sequence list may contain one or more of the additional volume set members, following the mandatory first volume. If the sequence list contains the entire volume set membership (which may be only one volume), it may then contain one or more volume set candidates. Volume set candidates can become volume set members only as the result of an output operation. When an output operation causes the amount of data in the file set to exceed the capacity of the current volume set membership, the first available volume set candidate becomes a volume set member.

When the first attachment to any file in a file set is made, the volume sequence list for the file set is initialized from the attach description. At detach time, the I/O module empirically determines that one or more volumes are volume set members, by virtue of having used them in the course of processing the attached file. The remaining volumes in the sequence list, if any, are considered to be candidates. In subsequent attachments to any file in the file set, the order of volumes specified in the attach description is compared with the sequence list. For those volumes that the I/O module knows to be volume set members, the orders must match; otherwise, an error is indicated. Those volumes in the sequence list that the I/O module considers to be candidates are replaced by attach description specifications, if the orders differ. If the attach description contains more volumes than the sequence list, the additional volumes are appended to the list. This implementation maintains and validates the volume set membership on a per-process basis, and maintains a list of volume set candidates that is alterable on a per-attach basis.

Once a volume sequence list exists, subsequent attachments to files in the file set do not require repeated specification of any but the first (or only) volume, which is used to identify the file set. If the I/O module detects physical end of tape in the course of an output operation, it prepares to switch to the next volume in the volume set. An attempt is made to obtain the volume name from the sequence list, either from the sublist of members, or the sublist of candidates. If the list of volume set members is exhausted, and the list of candidates is either empty or exhausted, the user is queried for permission to terminate processing. If the reply is negative, the I/O module queries for the volume name of the next volume, which becomes a volume set member and is appended to the volume sequence list. If a volume name is obtained by either method, ✱ volume switching occurs, and processing of the file continues.

If the I/O module reaches end-of-file section (but not of file) in the course of an input operation, it first attempts to obtain the next volume name from the volume sequence list. No distinction is made between the member and candidate sublists, because a volume that ends with a file section must be followed by the volume that contains the next section. If the sequence list is exhausted, the user is queried as described above. If either of these methods results in a volume name, volume switching occurs and processing of the file continues.

If the volume set is demounted at detach time, all volume set candidates ✱ are purged from the volume sequence list.

tape_ibm_

Multiple Devices

If a volume set consists of more than one volume, the -device 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 file or any other in the file set is N equals the previous value of N.

File Set Density

The I/O module makes provision for three densities: 800, 1600, and 6250 bpi (bits per inch). Every file in a file set must be recorded at the same density; otherwise, an error is indicated.

The -density control argument is used to explicitly specify the file set density, where N specifies the density at which the file set is (to be) recorded (N can be 800, 1600, and 6250 bpi). The file set density can only be changed in a subsequent attachment if the volume set was demounted by the previous attach.

In the absence of a -density control argument, the file set density is determined as follows:

open for input: N = density of VOL1 label
open for output, creating new file set: N = 1600 bpi
open for output, old file set: N = density of VOL1 label

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.

Opening

The opening modes supported are sequential_input and sequential_output. An I/O switch can be opened and closed any number of times in the course of a single attachment. Such a series of openings may be in either or both modes, in any valid order.

All openings during a single attachment are governed by the same attach description. The following control arguments, all of which pertain to output operations, are ignored when the switch is opened for sequential_input:

-create -force
-expires -modify
-extend -replace

Resource Disposition

The 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.

If several attaches and detaches to a file set are made in a process, repeated assignment and unassignment of resources is undesirable. Although the processing time required to assign and unassign a device is small, all available devices can be assigned to other processes in the interval between one detach and the next attach. While volumes are not often "competed" for, mounting and demounting is both time-consuming and expensive.

The -retain control argument is used to specify retention of resources across attachments, where STR specifies the detach-time resource disposition. If STR is the string all, all devices and volumes remain assigned to the process. If STR is the string none, all devices and volumes are unassigned. This is the default retention.

The I/O module provides a further means for specifying or changing the resource disposition subsequent to attachment. If retention of any devices or volumes has been specified at or subsequent to attach time using the retention control operation, the unassign_resource command cannot be used. Instead, use the retain_none or retention -none control operation before detaching the I/O module. (See "retention, retain_none, retain_all Operations" under "Control Operations" below.)

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.

tape_ibm_

If the attach description contains any of the output control arguments (-extend, -modify, or -create), volumes are mounted with rings; otherwise, they are mounted without rings. When a volume set mounted with rings is opened for sequential_input, hardware file protect is used to inhibit any spurious write operations. A volume set mounted without rings cannot be opened for sequential_output.

However, the following sequence of events is possible. An attach description contains none of the output control arguments, but does contain the "-retain all" control argument. The volume set is mounted without rings. After one or more (or no) openings for sequential_input, the I/O switch is detached. The volume set remains mounted because of the "-retain all" control argument. Subsequently, an attach is made whose description contains an output control argument, which requires that the volume set be mounted with rings. However, as rings can only be inserted in a demounted volume, the entire volume set must be demounted and then remounted.

This situation can be avoided by using the -ring (-rg) control argument to specify that the volume set be mounted with write rings. If no output control argument is specified in conjunction with -ring, the I/O switch cannot be opened for sequential_output.

When a volume set is mounted with write rings and the I/O switch is opened for sequential_input, the hardware file protect feature is used to safeguard the file set.

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 (described in the MPM Subroutines). The user may intercept one or more types of query by establishing a handler for the command_question condition, which 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 sequential_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 is probably 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 sequential_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 sequential_output.

query_code = 2 — the first block of the tape is not a valid IBM VOL1 label. The tape is not formatted as an IBM SL volume. This query code can occur only if the I/O stream is opened for sequential_output.

query_code = 3 — the volume identifier recorded in the VOL1 label is incorrect. The volume identifier does not match the volume name.

query_code = 11 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 sequential_output.

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_$unexpired_file .

This can occur only when the I/O switch is open for sequential_output. A file that must be extended, modified, 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 sequential_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.)

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 sequential_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 is probably 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 sequential_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 sequential_output.

query_code = 2 — the first block of the tape is not a valid IBM VOL1 label. The tape is not formatted as an IBM SL volume. This query code can occur only if the I/O stream is opened for sequential_output.

query_code = 3 — the volume identifier recorded in the VOL1 label is incorrect. The volume identifier does not match the volume name.

tape_ibm_

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 must be a volume name 6 characters or less in length, 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.

Structure Attribute Defaults

When a file is created, the I/O module can supply a default value for any or all of the file structure attributes. The defaults used are as follows:

1. record format - the default is f = vb
2. block length - the default is b = 8192
3. record length f = u: undefined
   f = fb | f: r = block length
   f = vb | v: r = block length - 4
   f = vbs _ vs: r = 1044480

An injudicious combination of explicit specifications and defaults can result in an invalid attribute set. For example, if -record 12000 is specified, applying the defaults produces the following:

-format vb -block 8192 -record 12000

This attribute set is invalid because, in vb format (see "Record Formats" below) the record length must be less than or equal to the block length minus 4.

Overriding Structure Attributes

Normally, the -format, -block, and -record control arguments are not included in the attach description of an I/O switch that is opened for sequential input; the structure attributes are extracted from the file labels. However, the I/O module permits the recorded structure attributes to be overridden by explicitly specified attach description control arguments. Because the apparent structure and characteristics of the file can be drastically altered, great care must be taken to ensure that attribute overrides do not produce unexpected and unwanted results.

If a file has the following recorded attributes:

-format fb -block 800 -record 80

an explicit specification of the -format fb and -record 800 control arguments causes each block of ten 80-character records to be treated as a single 800-character record.

If a file has the following recorded attributes:

-format fb -block 800 -record 80

an explicit specification of the -format fb, -block 80, and -record 80 control arguments causes the last 720 characters of every block to be discarded. No error is indicated, because every block of the file contains at least one 80-character record.

Record Formats

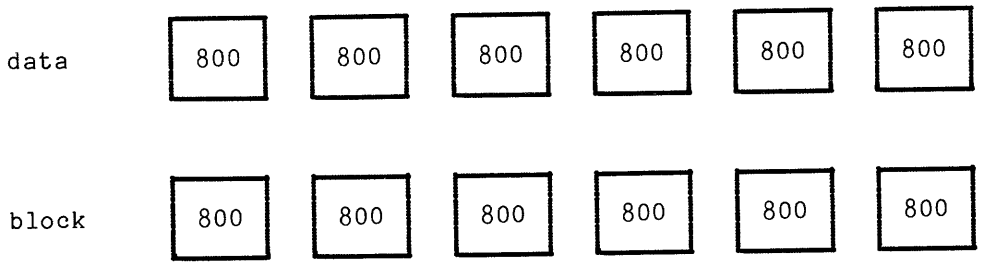
Files are structured in one of four record formats: F(B), V(B), V(B)S, or U. When a file is created, its record format should be chosen in accordance with the nature of the data to be recorded. For example, data consisting of 80-character card images is most economically recorded in FB format, blocked fixed-length records. Data consisting of variable length text lines, such as PL/I source code produced by a text editor, is best recorded in VBS format, blocked spanned records, so that blanks are not inserted except after the last line.

With the exception of U format, files are either blocked or unblocked, blocked being the usual case. Each block of an unblocked file contains just one record, whereas each block of a blocked file can contain several records. Blocking can provide a significant savings of processing time, because several records are accessed with a single physical tape movement. Furthermore, as blocks are separated by distances of blank tape, blocking reduces the amount of tape needed to contain a file.

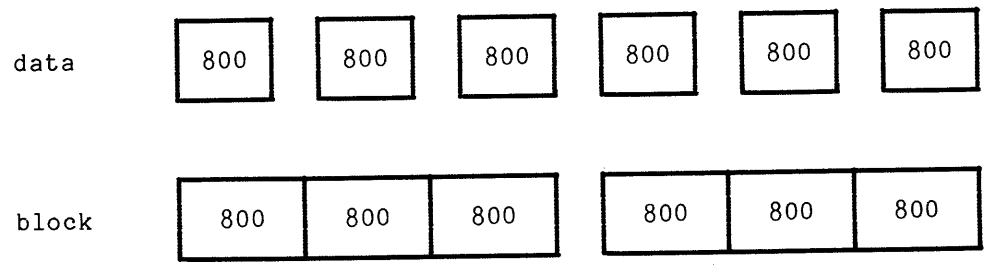
F(B) FORMAT

In F format, records are of fixed (and equal) length, and files have an integral number (N) of records per block. If the file is unblocked, N equals 1 and the record length (r) equals the block length (b). If the file is blocked, N > 1 and b equals (r * N) where N is known as the blocking factor.

For example, if r equals 800 and b equals 800, then the file is unblocked and each block contains just one record.



If r equals 800 and b equals 2400, then the file is blocked, the blocking factor is 3, and each block contains three records.



The Standard for F format records permits recording short blocks. A short block is a block that contains fewer than N records, when N is greater than 1. Although the I/O module can read this variant of F format, it writes a short block in only one case. The last block of a blocked file can contain fewer than N records if there are no more records to be written when the file is closed. Therefore, blocked F format files written by the I/O module are always in FBS (fixed blocked standard) format.

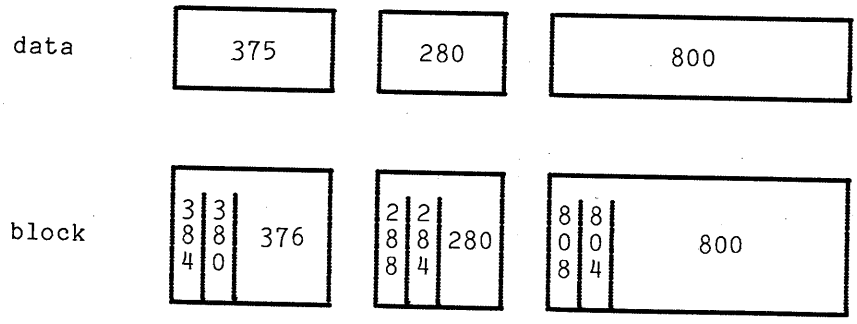
There are two special cases in which a datum is padded out to length r. The first case is that of iobl (the number of characters to be written) equals 0: a record of r blanks is written. When such a record is subsequently read, it is interpreted as a record of r blanks, and Not as a zero-length record. The second case is that of 0 is less than iobl is less than r: the record is padded on the right with blanks to length r, and the padded record written. When such a record is read, the original characters plus the padding are returned. The case of iobl is greater than r is in error.

V(B) FORMAT

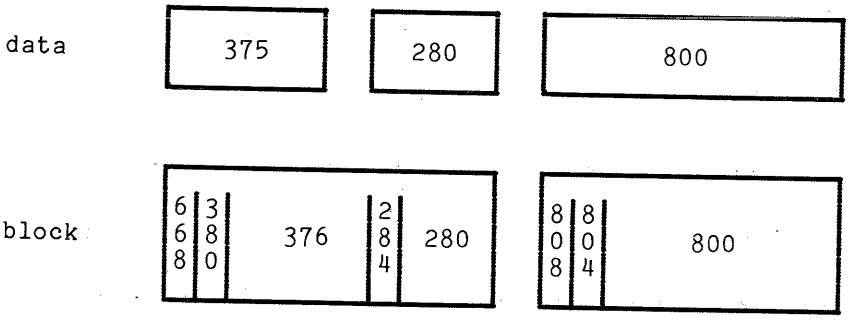
In V format, records and therefore blocks may vary in length. Each record is preceded by a four-character record descriptor word (RDW) that contains the actual record length in binary, including the length of the RDW itself. Each block is preceded by a four-character block descriptor word (BDW) that contains the actual block length in binary, including the length of the BDW itself.

V format files have an integral number of records per block, N. If the file is unblocked, b = r + 4; if blocked, b ≥ r + 4; For blocked records, the number of records per block varies indirectly with the size of the records.

If r equals 804, b equals 808, and the file is unblocked, records of up to 800 characters can be written, but each block can contain only one record.



If r equals 804, b equals 808, and the file is blocked, records of up to 800 characters can be written. Each block can contain a maximum of 201 zero-length records (a record written as a 4-character RDW containing the binary value 4).



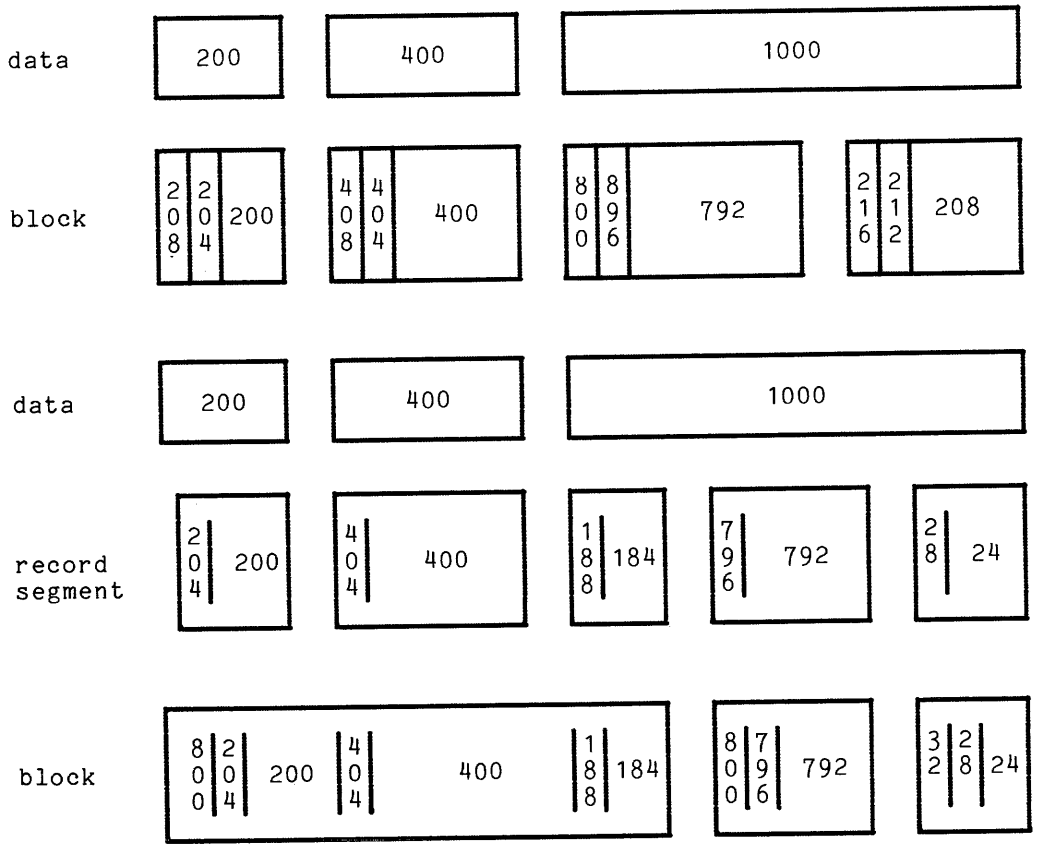
V(B)S FORMAT

In V(B)S format, a single record is formatted as one or more record segments. A record segment contains either a complete record, the initial portion of a record, a medial portion of a record, or the final portion of a record. No two segments of the same record can be contained in the same block, but a block may contain the segments of several different records. The maximum record length is limited only by the maximum size of a storage system segment, currently 1,044,480 characters.

V(B)S format files have an integral number of record segments per block. If the file is unblocked, each block contains only one record segment; if blocked, the number of record segments per block is variable. In either case, r and b are independent of one another.

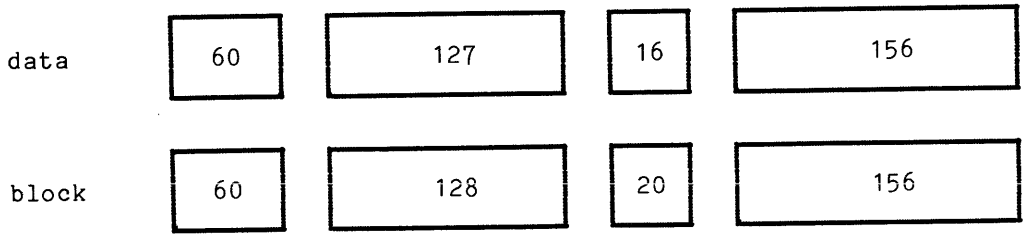
tape_ibm_

Each record segment begins with a four-character segment descriptor word (SDW). The SDW contains a four-character record segment length in binary, that includes the length of the SDW itself. (See "DOS Files" above.) The SDW also contains a one-character record segment code in binary, that indicates if the segment contains a complete record, or an initial, medial, or final portion. In the examples below, r equals 1000 and b equals 800.



U FORMAT

U format files contain records that do not conform to either F(B), V(B), or V(B)S format. A U format file is always unblocked. The record length is undefined, and the block length must equal or exceed the maximum record length. Blocks may vary in length. The special case of writing a record of less than 20 characters produces a block padded to length 20 with blanks.



tape_ibm_

Volume Initialization

The Standard requires that all volumes be initialized with VOL1 and dummy HDR1 labels before they are used for output. The I/O module provides a semiautomatic volume initialization mechanism that performs this operation as an integral part of the output function. It should be noted that, as stated above, a newly initialized volume contains a dummy HDR1 label, but not a dummy file. If a file is created on a newly initialized volume without an explicit specification of the -number control argument, the I/O module attempts to append it to the file set, resulting in an error.

Conformance To Standard

With two exceptions, the I/O module conforms to the Standard: the I/O module cannot process block lengths in excess of 8192 characters; and the I/O module ignores the data set security field in the HDR1 label on input, and records it as 0 on output.

Label Processing

VOL1

The label is processed on input and output. The owner-name and address-code-field, character positions (CP) 42 to 51, holds a three-character volume authentication code.

UVL1 - UVL8

These labels are not written on output and ignored on input.

HDR1/EOF1/EOV1

The labels are processed on input and output. The system-code-field, CP 61 to 73, is recorded as "MULTICS IBM ".

HDR2/EOF2/EOV2

The labels are processed on input and output. The 17-character job/job-step-identification-field, CP 18 to 34, is recorded as follows:

"MULTICS /" || Julian creation date || " "

HDR3/EOF3/EOV3 - HDR8/EOF8/EOV8

These labels are not written on output and are ignored on input.

These labels are not written on output and are ignored on input.

UHL1/UTL1 - UHL8/UTL8

These labels are not written on output and are ignored on input.

Error Processing

If an error occurs while reading, the I/O module makes 25 attempts to backspace and reread. If an error occurs while writing, the I/O module makes 10 attempts to backspace, erase, and rewrite. Should an error while reading or writing data prove to be unrecoverable, the I/O Module "locks" the file, and no further I/O is possible. (See reset_error_lock OPERATION, below.) 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" above.) 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

Close Operation

The I/O switch must be open.

Control Operation

The I/O module supports eleven control operations.

hardware_status	close_rewind
status	retention
	retain_none
	retain_all
volume_status	reset_error_locke
file_status	volume_density
feov	

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

hardware_status OPERATION

This operation returns the 72-bit IOM status string generated by the last tape I/O operation. The I/O switch must be open. The substr argument (IOM_bits, 3, 10) contains the major and minor status codes generated by the tape subsystem itself. (See MTS500 Magnetic Tape Subsystem, Order no. DB28 for an explanation of major and minor status.) The variable to which info_ptr points is declared as follows:

```
declare IOM_bits bit(72) aligned;
```

status OPERATION

This operation returns a structure that contains an array of status codes, providing an interpretation of the IOM status string generated by the last tape I/O operation. These codes may be used in calls to the com_err_ subroutine, or may be converted to printable strings by calling the convert_status_code_ subroutine. (See the description of the convert_status_code_ subroutine in the MPM Subsystem Writers' Guide and the description of the com_err_ subroutine in the MPM Subroutines.) The I/O switch must be open. The structure to which info_ptr points, device_status.incl.pl1, is declared as follows:

```
dcl  dstat_ptr          pointer;
dcl 1 device_status     based (dstat_ptr),
    2 IOM_bits          bit(72) aligned,     /* IOM status */
    2 n_minor           fixed bin,           /* number of minor codes */
    2 major             fixed bin(35),       /* major status code */
    2 minor             (10) fixed bin(35); /* minor status codes */
```

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. If the switch was closed by the I/O module as the result of an error while writing file header labels, trailer labels, or tapemarks, the current volume is the last (or only) volume in the volume set. The structure to which info_ptr points, tape_volume_status.incl.pl1, is declared as follows:

```
dcl  tvstat_ptr          pointer;
dcl 1 tape_volume_status based (tvstat_ptr),
    2 volume_name        char(6),    /* volume name */
    2 volume_id          char(6),    /* from VOL1 label */
    2 volume_seq         fixed bin,  /* order in volume set */
    2 tape_drive         char(8),    /* tape drive name */
                                     /* "" if not mounted */
    2 read_errors        fixed bin,  /* read error count */
    2 write_errors       fixed bin;  /* write error count */
```

In the current implementation of the I/O module, read_errors and write_errors are always zero. Eventually, the resource control package (RCP) supplies these values.

file_status OPERATION

This operation returns a structure that contains the current status of the file specified in the attach description. If the I/O switch has never been opened, no information can be returned; this situation is indicated by tape_file_status.state = 0. If the switch was opened, but is now closed, the current status of the file is its status just prior to closing. If the switch was closed by the I/O module as the result of an error while writing file header labels, trailer labels, or tapemarks, the entire file may have been deleted. In this case, the structure contains the current status of the previous file in the file set, if any. The structure to which info_ptr points, file_status.incl.pl1, is declared as follows:

```
dcl  tfstat_ptr          pointer;
dcl 1 tape_file_status   based (tfstat_ptr),
    2 state              fixed bin,      /* 0 - no information */
                                         /* 1 - not open */
                                         /* 2 - open, no events */
                                         /* 3 - open, event lock */
    2 event_code         fixed bin(35),  /* error_table_ code if
                                            state = 3 */
    2 file_id            char(17),       /* file identifier */
                                         /* "" if -no_labels */
    2 file_seq           fixed bin,      /* order in file set */
    2 cur_section        fixed bin,      /* current or last
                                            section processed */
    2 cur_volume         char(6),        /* volume name of volume
                                            on which cur_section
                                            resides */
    2 pad1               fixed bin,      /* not used */
    2 pad2               fixed bin,      /* not used */
    2 creation           char(5),        /* Julian creation date */
                                         /* "00000" if -no_labels */
    2 expiration         char(5),        /* Julian expiration date */
                                         /* "00000" if -no_labels */
    2 format_code        fixed bin,      /* 1 - U format */
                                         /* 2 - F(B) format */
                                         /* 3 - V(B) format */
                                         /* 4 - V(B)S format */
    2 blklen             fixed bin,      /* block length */
    2 reclen             fixed bin(21),  /* record length */
    2 blocked            bit(1),         /* "0"b - no | "1"b - yes */
    2 mode               fixed bin,      /* 1 - ASCII */
                                         /* 2 - EBCDIC */
    2 cur_blkcnt         fixed bin(35);  /* current block count */
```

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

feov OPERATION

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

close_rewind OPERATION

This operation specifies that the current volume is to be rewound when the I/O switch is next closed. info_ptr should be a null pointer. The switch need not be open when the operation is issued. The operation effects only one close; subsequent closings require additional control calls.

retention, retain_none, retain_all OPERATIONS

These operations cause the tape resources currently in use, i.e., tape drives(s) and tape volume(s), to be unassigned or retained at detach time according to the specified retention argument or operation. The info_ptr points to a fixed binary number with value as defined below:

1 retention -none or retain_none
causes none of the tape resources currently in use to remain assigned at detach time.

2 retention -volume
causes the tape volume(s) currently in use to remain assigned at detach time.

3 retention -device
causes the tape drives(s) currently in use to remain assigned at detach time.

4 retention -all or retain_all
causes all of the devices and volumes currently in use to remain assigned at detach time.

reset_error_lock OPERATION

This operation unlocks the files so that further I/O is possible subsequent to a parity-type I/O error while reading. Such an error is indicated by a previous iox_$read_record or iox_$position call having returned the status code error_table_$tape_error. In this case, the value of tape_file_status.event_lock is error_table_$tape_error. (See file_status OPERATION, above.) The I/O switch must be open for sequential_input. The info_ptr should be a null pointer.

tape_ibm_

NOTE: IF RECORDS ARE BLOCKED AND/OR SPANNED, THE VALIDITY OF ANY RECORDS READ SUBSEQUENT TO A PARITY-TYPE I/O ERROR IS NOT GUARANTEED. (The parity error is reported for the first read of a logical record in the block. The actual location of the error in the block in unknown.)

volume_density OPERATION

This operation returns the encoded density of the volume set. The I/O switch need not be open. The variable to which info_ptr points is declared as follows:

```
declare volume_density fixed bin;
```

The values returned and their meanings are listed below:

value	meaning
-1	none specified yet
2	800
3	1600
4	6250

Detach Operation

The I/O switch must be closed. If the I/O module determines that the membership of the volume set may have changed, the volume set members are listed before the set is demounted; volumes not listed are available for incorporation into other volume sets. If the volume set is unlabeled, only the name of the last volume processed is listed.

Modes Operation

This I/O module does not support the modes operation.

Position Operation

The I/O switch must be open for sequential_input. The I/O module does not support skipping backwards. In the course of a position operation, events or errors may occur that invoke the query mechanism. (See "Queries" above.) An unrecoverable error locks the file, and a severe error causes the I/O module to close the I/O switch.

Read Length Operation

The I/O switch must be open for sequential_input. In the course of a read_length operation, events or errors may occur that invoke the query mechanism. (See "Queries" above.) An unrecoverable error locks the file, and a severe error causes the I/O module to close the I/O switch.

Read Record Operation

The I/O switch must be open for sequential_input.

Write Record Operation

The I/O switch must be open for sequential_output.

Unlabeled Tapes

The I/O module supports basic processing of unlabeled tapes that are structured according to the OS Tape Labels document mentioned at the beginning of this description. DOS leading tape mark (LTM) unlabeled format tapes cannot be processed.

The -no_labels control argument specifies that unlabeled tapes are to be processed. The -no_labels control argument and any of the following control arguments are mutually exclusive:

-name	-extend
-replace	-modify
-expires	-dos
-force	

Volume switching is handled somewhat differently for unlabeled tapes. When the I/O module detects a tape mark in the course of an input operation, it determines whether or not any volumes remain in the volume sequence list. If another volume appears in the list, volume switching occurs and processing continues on the next volume. If the list is exhausted, the I/O module assumes that end of information has been reached. Detection of end of tape during an output operation is handled in much the same way as it would be for a labeled tape. (See the OS Tape Labels document for a complete description of unlabeled volume switching strategy.)

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

tape_ibm_

where:

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

2. operation
   is any of the control operations previously described and summarized below.

operation	abbreviation	control_arg
status	st	-all
hardware_status	hst	
reset_error_lock	rel	
file_status	fst	
volume_status	vst	
retention	ret	-none, -volume, -device, -all
retain_all	reta	
retain_none	retn	
close_rewind	crw	
feov	feov	

3. control_arg
   is an operation control argument valid only for the retention and the status operations. A control argument is required for the retention operation; possible control arguments are described below:

   -none
   causes none of the tape resources currently in use to remain assigned at detach time.

   -volume
   causes the tape volume(s) currently in use to remain assigned at detach time.

   -device
   causes the tape drives(s) currently in use to remain assigned at detach time.

   -all
   causes all of the devices and volumes currently in use to remain assigned at detach time.

   The -all control argument is optional for the status operation. This control argument prints all available status information such as the device status, the volume status, the file status, and the hardware status. The -all control argument is only for use with the status operation through the io_call command. It is not defined for use in the status operation with iox_$control directly.

Examples

In the following examples, it must be emphasized that an attach description describes a potential operation, and in and of itself does nothing to the file. Depending upon the sequence of openings in various modes, one attach description can perform diverse functions.

```
tape_ibm_ 042381 -nm ARD21 -cr -fmt vbs -ret all
```

A file named ARD21 is to be appended to the file set whose first volume is 042381. If a file named ARD21 already exists in the file set, openings for sequential_input access that file, and openings for sequential_output replace the old file of that name. If no file named ARD21 already exists in the file set, openings for sequential_input prior to the first opening for sequential_output fail. The first opening for sequential_output creates the file by appending it to the end of the file set. Subsequent openings for sequential_input access the newly created file, and subsequent openings for sequential_output replace it. Spanned records are specified; the block length defaults to 8192, the record length to 1044480, and the encoding mode to EBCDIC. The density defaults to 1600 cpi, and the maximum number of devices defaults to 1. The volume set and devices are retained after detachment.

```
tape_ibm_ 042381 -nm fargo.pl1 -nb 2 -cr -force -fmt fb -bk 800 -rec 80
```

A file named fargo.pl1 is created at position 2 in the file set. If a file named fargo.pl1 already exists at position 2, openings for sequential_input prior to the first opening for sequential_output access that file. The first opening for sequential_output creates a new file, and subsequent openings for sequential_input access the new file. If no file named fargo.pl1 exists at position 2, openings for sequential_input prior to the first opening for sequential_output fail. If a file exists at position 2, it is replaced irrespective of its expiration date.

```
tape_ibm_ 042381 -nm zbx -rpl zbx -cr -md ascii -bk 6000 -exp 2weeks
```

A file named zbx is created, replacing a file of the same name. Openings for sequential_input prior to the first opening for sequential_output access the old file. Each opening for sequential_output creates a new file, and each subsequent opening for sequential_input access the most recently created file. The specified encoding mode is ascii. The record format defaults to VB, and the record length defaults to 5996 because the block length is specified as 6000. The file is protected from overwriting for a period of two weeks, so each opening for sequential_output subsequent to the initial opening for sequential_output causes the user to be queried for permission to overwrite.

```
tape_ibm_ 042381 042382 -nb 14 -nlb -cr -dv 3
```

A file is to be created at position 14 on volume 042381. If a file already exists at position 14, an opening for sequential_input prior to the first opening for sequential_output accesses that file; otherwise, an error is indicated. Openings for sequential_output create new files, and openings for sequential_input subsequent to the first opening for sequential_output access the most recent creation. The default record format is VBS, the default block length 8192, and the default record length 1044480. The volume set is unlabeled. If the file exceeds the capacity of volume 042381, it is continued on volume 042382. If it then exceeds the capacity of volume 042382, the user is queried for instructions. A maximum of three devices can be used.

```
tape_ibm_ 042381 042382 042383 -nm THESIS -ring
```

tape_ibm_

A file named THESIS is to be read. The I/O switch can only be open for sequential_input. The volume set consists of at least three volumes, and they are mounted with write rings. Only one device can be used.

```
tape_ibm_ 042381 -nm FF -nb 3 -ext -dv 4 -ret all
```

A file named FF at position 3 in the file set is to be extended. Each opening for sequential_input accesses the current version. Each opening for sequential_output produces a new version. A maximum of four devices can be used. Resources are retained after detachment.

```
tape_ibm_ 042381 -vol -COS -com in_slot_000034 -nb 6 -mod -fc
```

The file at position 6 in the file set is to be modified, irrespective of its expiration date. Each opening for sequential_input accesses the current version. Each opening for sequential_output produces a new version. The second volume of the volume set has volume identifier -COS, and can be found in slot 000034.

Attach Control Arguments

The following is a complete list of all valid attach control arguments in both long and short forms:

-block b	-bk b	20 ≤ b ≤ 32760 mod (b,4) = 0 if open for sequential_output
-clear	-cl	
-create	-cr	
-density N	-den N	N = 800 ¦ 1600 ¦ 6250
-device N	-dv N	1 ≤ N ≤ 63
-dos		
-expires date	-exp date	valid date
-extend	-ext	
-force	-fc	
-format f	-fmt f	f = fb ¦ f ¦ vb ¦ v vbs ¦ vs ¦ u
-mode STR	-md STR	STR = ebcdic ¦ ascii ¦ binary
-modify	-mod	
-name STR	-nm STR	STR ≤ 17 characters ≤ 8 characters (restricted subset) with -create
-no_labels	-nlb	
-number N	-nb N	1 ≤ N ≤ 9999
-record r	-rec r	1 ≤ r ≤ 1044480
-replace STR	-rpl STR	STR ≤ 17 characters
-retain STR	-ret STR	STR = all ¦ none
-ring	-rg	

The following is a list of positional keywords:

-comment STR	-com STR	STR ≤ 64 characters
-volume vni	-vol vni	volume name ≤ 6 characters

Name: tape_mult_

The tape_mult_ I/O module supports I/O to and from Multics standard tapes.

Usage

tape_mult_ reelid {-control_args}

where:

1. reelid
   is the name of the tape reel to be mounted for this attachment.

2. control_args
   can be chosen from the following:

   -comment STR, -com STR
   specifies a comment string that is displayed to the operator. It can be used to give the operator any special instructions that are relevant to this attachment. The comment string must be enclosed within quotes if it contains blanks or other spacing characters.

   -density N, -den N
   specifies the density setting of the attached tape drive, where N can be 800, 1600, or 6250 bpi. The defaults are 800 for 7-track, and 1600 for 9-track. When opened for reading, the specified density is used only as a first guess. If the tape cannot be read at that density, tape_mult_ tries the other density.

   -error_tally, -et
   when opened for stream_input, displays an error summary on the user_output stream upon closing the tape I/O switch. This error summary includes: total number of read errors; number of errors that were successfully recovered for each of 1 to 10 backspace/re-read retrys; number of errors that could not be recovered by backspace/re-reading but were successfully recovered by reading forward and finding a good copy of the original record in error; and the number of times that both backspace/re-read and read forward recovery failed, but successful recovery was accomplished by backspacing two files, forward-spacing two files (thus positioning the tape at the beginning of the current file after tape motion past the tape cleaner and head in both directions dislodges any buildup of oxide particles on the tape or head surface) and then reading forward until original record in error was read successfully. This information is obtained from metering data kept in the tape_mult_ work segment, defined by tmdb.incl.pl1.

   -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. The default is 9.

-write, -wrt
mounts the tape reel with a write ring. The default is to mount the tape reel without a write ring.

-system, -sys
increases tape performance by using more I/O buffers and other performance optimizations. Access to >system_control_1>rcp>workspace.acs or rcp_sys_ is required to use this control argument.

-volume_set_name STR, -vsn STR
specifies the contents of the volume set name field located in the tape label record (see section 3 of this manual for a description of the standard Multics tape label record). When opened for writing, STR is written into the volume_set_id field of the tape label record. If this control argument is not specified, the volume_set_id field will be set to blanks. When opened for reading, the volume_set_id field of the tape label is compared to STR. If they match or if the volume_set_id field is padded with blanks, the open operation is allowed to be completed. If the volume_set_id field and STR do not match and the volume_set_id is not padded with blanks, error_table_$bad_label is returned. STR can be up to 32 characters in length.

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.

Opening

The opening modes supported by tape_mult_ are stream_input and stream_output. The extend option is not allowed. If the opening mode is stream_output, the attach description must have specified the -write control argument.

Read Record Operation

The get_chars operation reads Multics standard records until either the caller's buffer is filled, or until the end of the tape volume is encountered. If not all the characters on a tape record fit into the caller's buffer, they are saved by the I/O module for the next get_chars call.

Write Record Operation

The put_chars operation formats the data into Multics standard records of 1024 data words each. Each record is written as it is filled. A partially filled record is not written onto the tape until it is filled with a subsequent put_chars operation, an error_count order is done, or the switch is closed.

Control Operation

The tape_mult_ I/O module supports the control operation with three orders.

error_count
This order is supported only for the stream_output opening mode. It causes all output currently buffered to be written. An up-to-date error count is returned in the (fixed bin) variable referenced by the info_ptr argument.

boot_program
This order allows the specification of a boot program to be written into the tape label record (see Section 3 for a discussion of the bootable Multics tape label record format and function). The specified boot program must be coded in absolute self-relocating ALM assembly language and must be less than or equal to 832 (1500 octal) locations in length. The specified boot program is overlayed starting at absolute location 300 (octal) in the tape label record. When a Multics tape containing a bootable label record is bootloaded, control is transferred to location 300 via the tape label record transfer vector, the first 8 words of a bootable Multics tape label record. The I/O switch must be closed when this control order is executed. The specified boot program is written onto the tape label record when the tape is subsequently opened for output. The info_ptr must point to a structure of the following form:

```
dcl  1 boot_program_info          based (info_ptr),
     2 version                    fixed bin,
     2 boot_program_ptr           pointer,
     2 boot_program_text_length   fixed bin (21),
     2 boot_program_name          char (32) unaligned;
```

where:

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

2. boot_program_ptr
   is a pointer to the beginning of the text section of the specified boot program.

3. boot_program_text_length
   is the length in 36-bit words of the text section of the specified boot program.

4. boot_program_name
   if nonblank, is the name of the boot program that the user wants recorded in the boot_pgm_path field of the label record. If boot_program_name is blank, then the absolute pathname of the boot program is written into the boot_pgm_path field of the label record.

get_boot_program
This order allows a boot program to be extracted from the tape label when the tape is opened for input. This control order must be issued immediately after the tape is opened for input and before the first read operation is begun. If it is executed later, then error_table_$no_operation is returned. The info_ptr must point to the boot_program_info structure defined above for the boot_program control order. The user must set the version number. Then a pointer to a buffer, containing the extracted boot_program, its length, and the entryname portion of the boot_program_pathname, is returned to the user. If the get_boot_program control order is executed on a tape that has a standard tape label record, boot_program_ptr is set to null.

Control Operations From Command Level

All control operations can be performed from the io_call command, as follows:

io_call control switch order_arg

where:

1. switch
   is the name of the I/O switch.

2. order_arg
   must be one of the following:

   error_count
   boot_program PATH
   get_boot_program

Operations Not Supported

The tape_mult_ I/O module does not support the following operations:

get_line
modes

Name: tape_nstd_

The tape_nstd_ I/O module supports I/O to/from tapes in nonstandard or unknown formats. This module makes no assumptions about the format of the tape and returns one logical record for each physical record on the tape. Since the information upon the tape, including tape marks, is not interpreted by this I/O module, the user must detect the logical end of information on the reel.

Entry points in the module are not called directly by users; rather, the module is accessed through the iox_ subroutine. See the MPM Reference Guide for a general description of the I/O system and for a discussion of files.

Attach Description

The attach description has the following form:

```
tape_nstd_ reel_num {-control_args}
```

where:

1. reel_num
   is the tape reel number.

2. control_args
   can be chosen from the following:

   -block N, -bk N
   specifies the maximum record length, in bytes, for this attachment. The default value for N is 11200. Values of N greater than 11888 require access to either the >system_library_1>rcp_sys_ gate or >sc1>rcp>workspace.acs (see "Buffer Size" below).

   -comment STR -com STR
   specifies a comment string that is displayed to the operator. It can be used to give the operator any special instructions that are relevant to this attachment. The comment string must be enclosed within quotes if it contains blanks or other spacing characters.

   -density N, -den N
   specifies the initial density to be used for this attachment. Acceptable values for N are 200, 556, 800, 1600 and 6250; the default is 800 bpi.

   -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
   means that the tape is N track. Acceptable values for N are 7 and 9. If no track argument is supplied then 9 track is assumed.

tape_nstd_

-write
means that the tape is to be mounted with a write ring. This argument must occur if the I/O switch is to be opened for output or input/output.

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.

Open Operation

The opening modes supported are sequential_input, sequential_output, and sequential_input_output. If an I/O switch attached via the tape_nstd_ I/O module is to be opened for output or input_output, the -write control argument must occur in the attach description.

Control Operation

The following control operations are implemented by this I/O module:

backspace_file
positions the tape before the file mark next encountered while rewinding the tape (if no file mark is encountered then the tape is left at load point).

backspace_record
positions the tape before the previous record on the tape (or file mark if the current record is preceded by a file mark).

bcd
sets hardware mode to binary coded decimal (BCD). See "Hardware Modes" below.

binary
sets hardware mode to binary (this is the default). See "Hardware Modes" below.

data_security_erase
erases the tape media from its current position to the end of tape (EOT) reflective marker. Additional "erase" control orders can be issued to erase any data written beyond the EOT reflective marker. No more than 40 additional erase control orders should be issued since the tape volume could run off the supply reel.

d200
sets density to 200 bpi.

d556
sets density to 556 bpi.

d800
sets density to 800 bpi. This is the default.

d1600
sets density to 1600 bpi.

d6250
sets density to 6250 bpi.

erase
erases tape for a distance of three inches from the current position.

fixed_record_length
specifies that no record length information is expected by the caller since all data records are of a fixed length specified by a fixed bin(21) value. The record length is specified in bytes.

forward_file
positions the tape past the next file mark encountered on the tape.

forward_record
positions the tape after the next record (or file mark if one follows the current record) encountered on the tape.

io_call
supports the io_call command protocol for orders that expect nonnull info pointers. This order is prepared to interpret and print the status returned by the saved_status and request_status orders.

nine
sets hardware mode to eight/nine bit conversion. See "Hardware Modes" below.

protect
sets write inhibit regardless of the presence of a write permit ring in the tape reel. The tape unit will remain write inhibited until the tape is detached.

request_status
interrogates the tape controller and returns its status as a bit(12) aligned quantity.

reset_status
causes all resettable statuses of the tape unit to be reset.

retry_count
specifies a fixed bin(17) value which is the number of times an operation is to be retried before returning an error to the caller. The default value for the retry count is 10.

rewind
rewinds the tape to load point.

saved_status
returns the last status returned from the tape controller as a bit(12) aligned quantity.

unload
rewinds the tape and unloads it (done automatically when the tape is detached).

tape_nstd_

write_eof
Writes an end of file mark (EOF).

Hardware Modes

In BCD mode, allowed only for 7-track drives, 6-bit characters are translated and then put on tape one character per frame. The translation is reversed on input.

In nine mode, on output four 8-bit bytes are written from each word ignoring the high order bit of each 9-bit byte (by truncating it). On input, 8-bit characters are converted to 9-bit characters by forcing the high order bit to zero (by appending a zero-bit). This mode should be used to put ASCII or EBCDIC data on tape for transfer to other systems with 8-bit bytes.

In binary mode, all 36 bits of each word are read or written. This mode should be used for native Multics applications where binary data is written to tape.

9-track write 9 8-bit bytes (2 word) are written to 9 frames on tape.
9-track read 9 frames are read into 9 8-bit bytes (2 words).

7-track write 6 6-bit frames from each word.
7-track read 6 frames on tape are read into 6 6-bit characters (1 word).

7-track is 6 data + 1 parity track.
9-track is 8 data + 1 parity track.

Modes Operation

This I/O module does not support the modes operation.

Position Operation

This I/O module does not support the position operation.

Read Length Operation

This I/O module does not support the read length operation.

tape_nstd_

Close Operation

The close operation rewinds the tape reel. The tape remains mounted, and positioned at the load point. No further I/O operations may be performed unless the I/O switch is opened again.

Detach Operation

The detach operation unloads the tape.

Read Record Operation

The logical record returned by the read_record operation contains m=ceil(n/36) words, where n is the number of data bits in the physical record. The first n bits of the input record are the data bits, the last m-n bits are 0's. The buffer supplied to the read_record operation must be word aligned. Read requests are retried 10 times before reporting an error unless a retry_count control order has been used to change the retry count.

Write Record Operation

The logical record supplied to the write_record operation must be word aligned, and must contain 0 mod 36 data bits.

Notes

This I/O module violates those iox_ conventions that seem ill suited to processing raw tapes. In particular, read record and skip record operations may pass file marks. For example, if a tape contains two records, A and B, separated by a file mark, then the first read request would read record A, a second read request would return error_table_$end_of_info, and a third read request would return record B.

Buffer Size

The maximum number of bytes that may be transmitted on a read_record or write_record operation is 180224, less overhead. This limit may be administratively restricted to a lower value. To use the full capability, the caller may need access to >system_library_1>rcp_sys_ or >sc1>rcp>workspace.acs.

SECTION 6

PROGRAMMING EXAMPLES

This section gives several examples of the use of the Multics peripheral I/O facilities. The writing out and subsequent reading in of a segment to and from magnetic tape is performed in several ways throughout this section.

USER-RING I/O SYSTEM COMMANDS

To write out a tape (for example, reel 50015) from the segment >udd>Work>Green>data, issue the following commands at Multics command level:

```
io_call attach tape_switch tape_ansi_ 50015 -name data
     -create -number 1 -fmt s

io_call open tape_switch sequential_output

io_call write tape_switch -segment >udd>Work>Green>data

io_call close tape_switch

io_call detach tape_switch
```

To read a tape back in again, to >udd>Work>Green>new_data, issue the following commands at Multics command level:

```
io_call attach tape_switch tape_ansi_ 50015 -name data

io_call open tape_switch sequential_input

io_call read tape_switch 1048576 -segment new_data

io_call close tape_switch

io_call detach tape_switch
```

For the meanings of the particular control arguments to the tape_ansi_ I/O module, see the description of this I/O module in Section 5 of this manual.

This sequence of calls writes and reads back an ANSI standard tape. A file named data, in spanned record format, is created on the tape and read back. The number, 1048576, is the maximum number of characters to be read. This number must be given to the io_call command on a read request. This value is the maximum number of characters in a segment. If the tape is not already in ANSI format, the tape_ansi_ I/O module queries the user if the tape is to be initialized to ANSI format. (The io_call command is described in the MPM Commands.)

This technique for performing tape I/O has the advantage that no programs need be written to use it. Simple commands, without the need for preparing control files, suffice. Using the abbrev or exec_com facilities, a segment can be written to tape in this manner with one command line. If the only need to be filled is that of storing a segment or several segments on tape, this method is completely adequate. The list_tape_contents command, described in Section 4, can be used to list the contents of the tape produced in this manner.

This method of utilizing tape has the obvious disadvantage that it is completely interactive. A facility that needs to deal with tape from program code cannot use this method.

PL/I CALLS TO THE USER-RING I/O SYSTEM

Figures 6-1 and 6-2 contain sample programs to write out the segment >udd>Work>Green>data to tape 50015 with the file name data and to read it in again.

These two PL/I programs are written to accomplish the same effect as the I/O system commands in the previous examples. Each call to the iox_ subroutine (documented in the MPM Subroutines) has the same effect as one call to the io_call command as used above. Each call to the iox_ subroutine returns an error code, represented in the programs by the value of the variable "code." In this case, the code variable always has the value of the status code returned by the tape_ansi_ I/O module. This value is tested at each point to check for error and report any problem that arises.

This technique for performing peripheral I/O has the advantage that a program can call all entries in the user-ring I/O system and all entries of a particular I/O module and, thus, perform all operations documented in the description of each I/O module. The optimum flexibility in user-ring I/O is achieved in this manner.

The disadvantages of this technique lie chiefly in the number of calls that must be made to the user-ring I/O system to attach and detach a device. To perform correct recovery, should any step fail or should a release be performed around the stack frame of the program (a contingency dealt with in "Language I/O in PL/I with Protocol-Defined Data Format" below), requires the setting of a number of switches to determine which calls must be undone. Furthermore, the entries of the user-ring I/O system are not callable from most languages other than PL/I.

```
iox_ansi_write: procedure;

/* This procedure writes out a segment to tape, using explicit calls to
   the user-ring I/O system to perform an attachment via tape_ansi_.  */


dcl  com_err_ entry options (variable);
dcl  iox_$attach_ioname entry (char (*), ptr, char (*), fixed bin (35));
dcl  iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35));
dcl  iox_$close entry (ptr, fixed bin (35));
dcl  iox_$detach_iocb entry (ptr, fixed bin (35));
dcl  iox_$write_record entry (ptr, ptr, fixed bin (21), fixed bin (35));

dcl  code fixed bin (35);
dcl  hcs_$initiate_count entry (char (*), char (*), char (*),
      fixed bin (24), fixed bin (1), ptr, fixed bin (35));
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));


dcl  p ptr;
dcl  iocbp ptr;                  /* Pointer value of switch tape_switch */
dcl  null builtin;
dcl  seg bit (bitcount) aligned based (p);
dcl  bitcount fixed bin (24);


          call hcs_$initiate_count
                    (">udd>Work>Green", "data",  "", bitcount, 0, p, code);
                                                  /* Get pointer to segment */
          if p = null
                    then call com_err_
                         (code, "iox_ansi_write", "Cannot initiate segment");
          else do;
               call iox_$attach_ioname ("tape_switch", iocbp,
               "tape_ansi_ 50015 -name data -create -number 1 -fmt s",
                    code);   /* Attach switch, mounting the tape */
               if code ^= 0 then call com_err_ (code, "iox_ansi_write",
                    "Cannot attach tape.");
               else do;
                    call iox_$open (iocbp, Sequential_output, "0"b, code);
                                       /* Open switch for stream output */
                    if code ^= 0 then
                         call com_err_ (code, "iox_ansi_write",
                              "Cannot open switch");
                    else do;
                         call iox_$write_record (iocbp, p, bitcount/9, code);
                                                  /* Write out data, integral
                                                     number of characters. */
                         if code ^= 0 then
                              call com_err_ (code, "iox_ansi_write",
                                   "Could not write data");
                         call iox_$close (iocbp, (0));  /* Close the switch. */
                    end;
                    call iox_$detach_iocb (iocbp, (0)); /* Demount the tape */
               end;

               call hcs_$terminate_noname (p, (0)); /* Clean up address space */
          end;
          return;


%include iox_modes;                        /* defines "Sequential_output" */
     end;
```

Figure 6-1. Writing Segment to Tape With PL/I Calls to iox_ (via tape_ansi_)

```
iox_ansi_read: procedure;

/* This procedure reads in a segment from tape, using explicit calls from
   the user-ring I/O system to perform an attachment via tape_ansi_.  */

dcl  error_table_$end_of_info fixed bin (35) external;
dcl  com_err_ entry options (variable);
dcl  iox_$attach_ioname entry (char (*), ptr, char (*), fixed bin (35));
dcl  iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35));
dcl  iox_$close entry (ptr, fixed bin (35));
dcl  iox_$detach_iocb entry (ptr, fixed bin (35));
dcl  iox_$read_record entry (ptr, ptr, fixed bin (21), fixed bin (21),
       fixed bin (35));

dcl  code fixed bin (35);
dcl  hcs_$make_seg entry (char (*), char (*), char (*),
     fixed bin (5), ptr, fixed bin (35));
dcl  hcs_$set_bc_seg entry (ptr, fixed bin (24), fixed bin (35));
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));


dcl  p ptr;
dcl  iocbp ptr;               /* Pointer value of switch tape_switch */
dcl  null builtin;
dcl  bitcount fixed bin (24);
dcl  char_count fixed bin (21);         /* Number of characters
                                           actually read. */


          call hcs_$make_seg (">udd>Work>Green", "new_data", "", 1010b, p, code);
                                                 /* Create new segment */
          if p = null then call com_err_ (code, "iox_ansi_read",
                    "Cannot make new segment");
          else do;
               call iox_$attach_ioname ("tape_switch", iocbp,
                    "tape_ansi_ 50015 -name data", code);
                                       /* Attach switch, mounting the tape */
               if code ^= 0 then call com_err_ (code, "iox_ansi_read",
                    "Cannot attach tape.");
               else do;
                    call iox_$open (iocbp, sequential_input, "0"b, code);
                                       /* Open switch for stream input */
                    if code ^= 0 then
                         call com_err_ (code, "iox_ansi_read",
                         "Cannot open switch");
                    else do;
                         call iox_$read_record (iocbp, p, 1048576, char_count,
                         code);
                                             /* Read in data, integral
                                                number of characters. */
                         if code ^= 0 & code ^= error_table_$end_of_info then
                                       /* We expect fewer than 1048576
                                          (4 * 2 ** 18) characters. */
                              call com_err_ (code, "iox_ansi_read",
                                   "Could not read data");
                         else do;
                              bitcount = char_count * 9; /* Compute bit count */
                              call hcs_$set_bc_seg (p, bitcount, code);
                              if code ^= 0 then
                                   call com_err_ (code, "iox_ansi_read",
                                   "Cannot set bit count to ^d.", bitcount);
                         end;
```

Figure 6-2. Reading Segment From Tape With PL/I Calls to iox_ (via tape_ansi_)

```
                    call iox_$close (iocbp, (0));  /* Close the switch. */
               end;
               call iox_$detach_iocb (iocbp, (0)); /* Demount the tape */
          end;

          call hcs_$terminate_noname (p, (0)); /* Clean up address space */
     end;
     return;

%include iox_modes;                    /* defines "sequential_input" */
     end;
```

Figure 6-2 (Cont). Reading Segment From Tape With PL/I Calls to iox_ (via tape_ansi_)

Figures 6-3 and 6-4 show sample programs to write and read a segment using the intrinsic I/O facilities of the PL/I language to access the tape via the tape_ansi_ I/O module.

The PL/I language I/O system makes all calls to the user-ring I/O system, including those to attach and detach the appropriate switch. The PL/I language provides no general way to obtain the length of a record that is read in. The environment (stringvalue) attribute can be used for this purpose, but this requires setting up a varying string, at least as large as the record to be read, and copying it. This is not always possible, since the record to be read in can be as large as a segment. Another way is to set up a record buffer as large as a segment and heuristically determine the length of the segment by "finding the end of it" via the adjust_bit_count_ subroutine (described in the MPM Subroutines). Obviously, this technique does not work for arbitrary binary data. (See "Language I/O in PL/I with Protocol-Defined Data Format" below for an alternative solution to this problem.) Since a record is being read whose length is not known, a record buffer (the variable "segment") is set up as having the length of a full-size segment. When PL/I reads the tape record, via a call to the tape_ansi_ module, a record shorter than this is read and the record condition is signalled. The "on record (tape);" on-unit in the reading program explicitly ignores this condition. Although standard PL/I does not define the contents of the buffer variable after a return from an on-unit for the record condition is performed, in this case Multics PL/I specifies that the record fill the low addresses of the buffer for the length of the record.

This technique has the advantage that no knowledge of I/O system calls is required. The meaning of PL/I statements that perform I/O is known to PL/I programmers on other systems, as opposed to calls to the Multics I/O system. The use of language I/O statements provides the fullest power of the language to the programmer using peripheral I/O.

The principal disadvantage of using language I/O is that not all calls accessible from the user-ring I/O system can be made via language I/O. In PL/I, for instance, no calls corresponding to "control" functions of I/O modules can be performed via language I/O. However, in certain circumstances the pl1_io_$get_iocb_ptr subroutine can be used to remedy this deficiency. See the MPM Subsystem Writers' Guide for a description of this entry point.) Thus, end-of-file marks, etc., cannot be written from language I/O. A more severe deficiency is the inability to determine the length of a record on tape. Via proper protocols, however, this deficiency too can be remedied. (See "Language I/O in PL/I with Protocol-Defined Data Format" below.)

```
tape_write: procedure;

dcl  p ptr;                        /* pointer on which segment image is based */
dcl  segment bit (bitcount) based (p) aligned;    /* image of the segment */
dcl  bitcount fixed bin (24);                     /* bit count of the segment */
dcl  hcs_$initiate_count entry (char (*), char (*), char (*), fixed bin (24),
          fixed bin (1), ptr, fixed bin (35));
dcl  code fixed bin (35);                                    /* status code */
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));
dcl  null builtin;
dcl  com_err_ entry options (variable);
                              /* used to report problem to error_output */

dcl  tape file internal;                          /* internal file for tape */

          call hcs_$initiate_count (">udd>Work>Green", "data", "", bitcount,
                    0, p, code);          /* get pointer and bit count */
          if p = null then do;                    /* could not initiate */
               call com_err_ (code, "tape_write", "Cannot get segment");
                                                  /* complain */
               return;
          end;
          open file (tape) sequential output title
                                    /* open the file, mount the tape */
               ("tape_ansi_ 50014 -nm data -cr -fmt s -nb 1");
                                     /* See description of tape_ansi_ */

          write file (tape) from (segment);
                                     /* Write the segment as a record */

          close file (tape);                      /* Demount the tape */
          call hcs_$terminate_noname (p, (0));
                                     /* Clean up address space of process */
          return;
     end;
```

Figure 6-3. Writing Segment to Tape With PL/I I/O Facilities

```
tape_read: procedure;

dcl  p ptr;                        /* pointer on which segment image is based */
dcl  record condition;
dcl  segment char (1048576) based (p) aligned;    /* image of the segment */
dcl  bitcount fixed bin (24);                     /* bit count of the segment */
dcl  adjust_bit_count_ entry (char (168) aligned, char (32) aligned,
          bit (1) aligned, fixed bin (24), fixed bin (35));
dcl  hcs_$make_seg entry (char (*), char (*), char (*), fixed bin (5),
          ptr, fixed bin (35));
dcl  code fixed bin (35);                                   /* status code */
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));
dcl  null builtin;
dcl  com_err_ entry options (variable);
                              /* used to report problem to error_output */

dcl  tape file internal;                          /* internal file for tape */

          call hcs_$make_seg (">udd>Work>Green", "new_data", "", 1010b, p, code);
          if p = null then do;                    /* could not initiate */
               call com_err_ (code, "tape_read", "Cannot make segment");
                                                  /* complain */
               return;
          end;
          on record (tape);   /* Ignore short records - Multics PL/I fills
                                 buffer anyway. */
          open file (tape) sequential input title
                              /* open the file, mount the tape */
               ("tape_ansi_ 50014 -nm data"); /* See description of tape_ansi_ */

          read file (tape) into (segment);

          close file (tape);                      /* Demount the tape */
          call adjust_bit_count_ (">udd>Work>Green", "new_data", "1"b,
                    bitcount, code);
          if code ^= 0 then
               call com_err_ (code, "tape_read",
                    "Cannot set bit count on new segment to ^d.", bitcount);
          call hcs_$terminate_noname (p, (0));
                                       /* Clean up address space of process */
          return;
     end;
```

Figure 6-4. Reading Segment to Tape With PL/I I/O Facilities

PROTOCOL-DEFINED DATA FORMAT

The programs in Figures 6-5 and 6-6 write out a segment to tape as a nonstandard tape and read it in again. Rather than being of Multics standard format (described in Section 3), the tape is written in a format known to, and used by, only these two programs.

A tape written in this nonstandard format contains segments written out in 100-word blocks, with the last fraction written as a short block. Each block occupies one physical tape record. The first record written on the tape is an image of the structure "hdr" in both programs, which is identical in both. Standard PL/I requires that the layout of the generation of storage from which a record is written be identical, or capable of being legally overlaid by, the layout of the generation of storage into which it is read. The variables in this record define how many 100-word records follow (they follow immediately after the header record), the length of the "short" record (zero if there is none), and the bit count to be set on the segment. No end-of-file mark is written: PL/I offers no facility for writing one, and none is necessary. The reading program knows only to read the header. When it has read the header, it knows, by convention with the writing program, exactly how many records, and of what sizes, to read.

The tape is blocked into 100-word records for several reasons. For one reason, the maximum size of any physical I/O buffer is limited by the tape controller and the other I/O hardware, as well as the software. Furthermore, the shorter a record is, the smaller is its chance of being written with errors. On the other hand, the larger the block size, the more efficient the use of the tape.

Part of the complexity of these programs stems from the fact that full error handling is attempted. The on-unit for "undefinedfile" is invoked if the tape cannot be attached for any reason. PL/I language I/O raises this condition if a call to the attach or open entry points of the tape_nstd_ I/O module fails. The transmit condition is raised if any error is indicated from any call that reads or writes data. There is no handler for the record condition, as it should never be raised. The length of all records is known by the reading program.

The entry pl1_io_$error_code (described in the MPM Subroutines) is used to extract the I/O system status code for use in error recovery. The PL/I language provides no intrinsic means to return error codes of the Multics I/O system.

A handler for the cleanup condition is provided in these programs to illustrate its use. Any program that performs I/O attachments should provide one. The cleanup handler closes the file tape, thus detaching the tape, should a release be performed around the invocation of the program.

These techniques have the advantage that the PL/I language can be used fully and within the language rules of PL/I, and without recourse to heuristics. These techniques can be used to define multivolume files, error recovery protocols, and other useful functions.

These techniques have the distinct disadvantage that a data format known only to the writer of such programs must be devised each time such an application is necessary. Such techniques are at cross ends with the goals of compatibility and standardization.

```
nstd_writer: procedure;

/* This program writes out a segment in the form of a nonstandard tape,
   in 100-word records, with a short last record if appropriate. A header
   record is written first, to tell the reading program how much
   to read, and how to set the bit count. */


dcl  tape file;

dcl (cleanup, transmit, undefinedfile) condition;
dcl  com_err_ external entry options (variable);  /* For errors */
dcl  hcs_$initiate_count entry (char (*), char (*), char (*),
     fixed bin (24), fixed bin (1), ptr, fixed bin (35));

dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));

dcl  pl1_io_$error_code entry (file) returns (fixed bin (35));
                                        /* Gets error code from file. */
dcl  p pointer;                         /* Pointer to segment, and
                                           to sliding "window" */

dcl  buffer (3) fixed bin (35);         /* Buffer for three-word record
                                           with bit and record counts. */

dcl 1 sliding_window based (p) aligned,          /* Moveable 100-word window
                                                    into segment. */
    2 data (100) fixed binary (35),              /* Real data words. */
    2 next_record fixed binary (35);    /* Beginning of next record. */

dcl 1 hdr aligned automatic,                     /* Header to be written. */
    2 bit_count fixed bin (24),                  /* Bit count of segment */
    2 word_count fixed bin (19),        /* Word count, for info only. */
    2 record_count fixed bin (17),               /* Full records written. */
    2 words_last_record fixed bin (17); /* Words in short last record. */


dcl short_last_record (hdr.words_last_record) fixed binary (35)
          based (p) aligned;


dcl  i fixed binary;
dcl  code fixed bin (35), null builtin;
```

Figure 6-5. Writing Segment to Nonstandard Tape

```
          call hcs_$initiate_count (">udd>Work>Green", "data", "", bit_count,
                    0, p, code);
                                                  /* Get pointer and length */
          if p = null then do;                    /* Can't get it. */
               call com_err_ (code, "nstd_writer", "Cannot initiate segment");
               return;
          end;

          on cleanup call clean_up_proc;          /* On abort, terminate
                                                     segment and close file */
          on undefinedfile (tape) call problem_report ("Cannot attach tape");
                                                  /* Set up for problem */
          on transmit (tape) call problem_report ("Transmission error on tape");

          open file (tape) title ("tape_nstd_ 50015 -write")
               sequential record output;   /* Attach the tape.*/

          hdr.word_count = (hdr.bit_count + 35)/36; /* Compute word length */
          hdr.record_count = hdr.word_count/100;  /* Find number of full, 100
                                                     word records */
          hdr.words_last_record = mod (hdr.word_count, 100); /* figure out
                                                     short record length. */

          write file (tape) from (hdr);           /* Write out the header. */

          do i = 1 to hdr.record_count; /* Write out all full records. */
               write file (tape) from (sliding_window.data);
                                                  /* Write 100 words */
               p = addr (sliding_window.next_record); /* Slide up the window. */
          end;

          if hdr.words_last_record ^= 0 then write file (tape)
                    from (short_last_record);
                                                  /* Write last record */

          call clean_up_proc;                     /* Detach and terminate. */
finish:   return;


/* Procedure to report problems */

problem_report: procedure (plaint);

dcl  plaint char (*);                             /* Specific message */

          call com_err_ (pl1_io_$error_code (tape), "nstd_writer", plaint);
          call clean_up_proc;                     /* Clean up. */
          go to finish;                           /* nonlocal exit */

end problem_report;

/* Procedure to clean up, detaching tape and terminating segment. */

clean_up_proc: procedure;

          call hcs_$terminate_noname (p, (0));

          close file (tape);                      /* It is permissible to
                                                     execute this, even if file
                                                       is not open. */

end clean_up_proc;

end nstd_writer;
```

Figure 6-5 (Cont). Writing Segment to Nonstandard Tape

```
nstd_reader: procedure;

/* This program reads in a segment written out as a nonstandard tape by the
sample writing-program, nstd_writer. It uses the header written by nstd_reader
to tell how many record and words to read. */


dcl  tape file;

dcl (cleanup, transmit, undefinedfile) condition;
dcl  com_err_ external entry options (variable);  /* For errors */
dcl  hcs_$set_bc_seg entry (ptr, fixed bin (24), fixed bin (35));
dcl  hcs_$make_seg entry (char (*), char (*), char (*),
     fixed bin (5), ptr, fixed bin (35));

dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));

dcl  pl1_io_$error_code entry (file) returns (fixed bin (35));
                                        /* Gets error code from file. */
dcl  p pointer;                         /* Pointer to segment, and
                                           to sliding "window" */

dcl  buffer (3) fixed bin (35);         /* Buffer for three-word record
                                           with bit and record counts. */

dcl 1 sliding_window based (p) aligned,          /* Moveable 100-word window
                                                    into segment. */
    2 data (100) fixed binary (35),              /* Real data words. */
    2 next_record fixed binary (35);    /* Beginning of next record. */

dcl 1 hdr aligned automatic,                     /* Header to be written. */
    2 bit_count fixed bin (24),                  /* Bit count of segment */
    2 word_count fixed bin (19),        /* Word count, for info only. */
    2 record_count fixed bin (17),               /* Full records written. */
    2 words_last_record fixed bin (17); /* Words in short last record. */


dcl short_last_record (hdr.words_last_record) fixed binary (35)
          based (p) aligned;


dcl  i fixed binary;
dcl  code fixed bin (35), null builtin;
```

Figure 6-6. Reading Segment to Nonstandard Tape

```
          call hcs_$make_seg (">udd>Work>Green", "new_data", "", 1010b, p, code);
                                                  /* Create new segment. */
          if p = null then do;                    /* Can't get it. */
               call com_err_ (code, "nstd_reader", "Cannot create segment");
               return;
          end;

          on cleanup call clean_up_proc;          /* On abort, terminate
                                                     segment and close file */
          on undefinedfile (tape) call problem_report ("Cannot attach tape");
                                                  /* Set up for problem */
          on transmit (tape) call problem_report ("Transmission error on tape");

          open file (tape) title ("tape_nstd_ 50015") /* Attach the tape. */
               sequential record input;

          read file (tape) into (hdr);            /* Read in header info. */

          do i = 1 to hdr.record_count;           /* Read in all full records. */
               read file (tape) into (sliding_window.data); /* Read 100 words. */
               p = addr (sliding_window.next_record); /* Slide up the window. */
          end;

          if hdr.words_last_record ^= 0 then read file (tape)
                    into (short_last_record);
                                                  /* Read last record */

          call hcs_$set_bc_seg (p, hdr.bit_count, code); /* Set the bit count. */
          if code ^= 0 then call com_err_ (code, "nstd_reader",
                    "Cannot set bit count to ^d", hdr.bit_count);
          call clean_up_proc;                     /* Detach and terminate. */
finish:   return;


/* Procedure to report problems */

problem_report: procedure (plaint);

dcl  plaint char (*);                             /* Specific message */

          call com_err_ (pl1_io_$error_code (tape), "nstd_reader", plaint);
          call clean_up_proc;                     /* Clean up. */
          go to finish;                           /* nonlocal exit */

end problem_report;

/* Procedure to clean up, detaching tape and terminating segment. */

clean_up_proc: procedure;

          call hcs_$terminate_noname (p, (0));

          close file (tape);                      /* It is permissible to
                                                     execute this, even if file
                                                       is not open. */

end clean_up_proc;

end nstd_reader;
```

Figure 6-6 (Cont). Reading Segment to Nonstandard Tape

The Multics standard tape format is a conventional format for writing arbitrary data on magnetic tape. The I/O module that implements this format, tape_mult_, is the only program that has, or need have, knowledge of this format. The Multics standard format provides for blocking and error recovery within it.

Figures 6-7 and 6-8 contain programs similar to those used in Figures 6-1 and 6-2 above, except that the tape_mult_ I/O module, using stream I/O, is used instead of the tape_ansi_ I/O module, with record I/O. By comparison, the attachment to tape_mult_ is substantially simpler to accomplish. On the other hand, since the Multics standard format does not provide for multiple files in one volume, or keeping name or generation information with data, the data on the tape is useless unless one knows what it represents. Thus, as opposed to the ANSI tape, the Multics standard tape is not self-identifying. There cannot be a command similar to list_tape_contents for this function.

The other advantages and disadvantages of this technique are those of the technique given in Figures 6-1 and 6-2 above.

There is no way to access a stream file of the form written by the tape_mult_ I/O module through PL/I language I/O.

```
tape_mult_writer: procedure;

/* This procedure writes out a segment to tape, using explicit calls to
   the user-ring I/O system to perform an attachment via tape_mult_.  */


dcl  com_err_ entry options (variable);
dcl  iox_$attach_ioname entry (char (*), ptr, char (*), fixed bin (35));
dcl  iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35));
dcl  iox_$close entry (ptr, fixed bin (35));
dcl  iox_$detach_iocb entry (ptr, fixed bin (35));
dcl  iox_$put_chars entry (ptr, ptr, fixed bin (21), fixed bin (35));

dcl  code fixed bin (35);
dcl  hcs_$initiate_count entry (char (*), char (*), char (*),
      fixed bin (24), fixed bin (1), ptr, fixed bin (35));
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));


dcl  p ptr;
dcl  iocbp ptr;                  /* Pointer value of switch tape_switch */
dcl  null builtin;
dcl  seg bit (bitcount) aligned based (p);
dcl  bitcount fixed bin (24);


          call hcs_$initiate_count (">udd>Work>Green", "data", "", bitcount,
               0, p, code);
                                                  /* Get pointer to segment */
          if p = null then call com_err_ (code, "tape_mult_writer",
               "Cannot initiate segment");
          else do;
               call iox_$attach_ioname ("tape_switch", iocbp,
               "tape_mult_ 50015 -write", code);
                         /* Attach switch, mounting the tape */
               if code ^= 0 then call com_err_ (code, "tape_mult_writer",
                    "Cannot attach tape.");
               else do;
                    call iox_$open (iocbp, Stream_output, "0"b, code);
                                    /* Open switch for stream output */
                    if code ^= 0 then
                         call com_err_ (code, "tape_mult_writer",
                              "Cannot open switch");
                    else do;
                         call iox_$put_chars (iocbp, p, bitcount/9, code);
                                              /* Write out data, integral
                                                 number of characters. */
                         if code ^= 0 then
                              call com_err_ (code, "tape_mult_writer",
                                   "Could not write data");
                         call iox_$close (iocbp, (0)); /* Close the switch. */
                    end;
                    call iox_$detach_iocb (iocbp, (0)); /* Demount the tape */
               end;

               call hcs_$terminate_noname (p, (0)); /* Clean up address space */
          end;
          return;


%include iox_modes;              /* defines "Stream_output" */
     end;
```

Figure 6-7. Writing Segment to Tape With PL/I Calls to iox_ (via tape_mult_)

```
tape_mult_reader: procedure;

/* This procedure reads in a segment from tape, using explicit calls from
   the user-ring I/O system to perform an attachment via tape_mult_.  */

dcl  error_table_$end_of_info fixed bin (35) external;
dcl  com_err_ entry options (variable);
dcl  iox_$attach_ioname entry (char (*), ptr, char (*), fixed bin (35));
dcl  iox_$open entry (ptr, fixed bin, bit (1) aligned, fixed bin (35));
dcl  iox_$close entry (ptr, fixed bin (35));
dcl  iox_$detach_iocb entry (ptr, fixed bin (35));
dcl  iox_$get_chars entry (ptr, ptr, fixed bin (21), fixed bin (21),
       fixed bin (35));

dcl  code fixed bin (35);
dcl  hcs_$make_seg entry (char (*), char (*), char (*),
     fixed bin (5), ptr, fixed bin (35));
dcl  hcs_$set_bc_seg entry (ptr, fixed bin (24), fixed bin (35));
dcl  hcs_$terminate_noname entry (ptr, fixed bin (35));


dcl  p ptr;
dcl  iocbp ptr;                /* Pointer value of switch tape_switch */
dcl  null builtin;
dcl  bitcount fixed bin (24);
dcl  char_count fixed bin (21);                    /* Number of characters
                                                      actually read. */


          call hcs_$make_seg (">udd>Work>Green", "new_data", "", 1010b,
               p, code);
                                                   /* Create new segment */
          if p = null then call com_err_ (code, "tape_mult_reader",
                    "Cannot make new segment");
          else do;
               call iox_$attach_ioname ("tape_switch", iocbp,
                    "tape_mult_ 50015", code);
                              /* Attach switch, mounting the tape */
               if code ^= 0 then call com_err_ (code, "tape_mult_reader",
                    "Cannot attach tape.");
               else do;
                    call iox_$open (iocbp, Stream_input, "0"b, code);
                              /* Open switch for stream input */
                    if code ^= 0 then
                         call com_err_ (code, "tape_mult_reader",
                         "Cannot open switch");
                    else do;
                         call iox_$get_chars (iocbp, p, 1048576,
                         char_count, code);
                                                   /* Read in data, integral
                                                      number of characters. */
                         if code ^= 0 & code ^= error_table_$end_of_info then
                                        /* We expect fewer than 1048576
                                           (4 * 2 ** 18) characters. */
                              call com_err_ (code, "tape_mult_reader",
                                        "Could not read data");
                         else do;
                              bitcount = char_count * 9; /* Compute bit count */
                              call hcs_$set_bc_seg (p, bitcount, code);
                              if code ^= 0 then call com_err_ (code,
                                   "tape_mult_reader",
                                   "Cannot set bit count to ^d.", bitcount);
                         end;
```

Figure 6-8. Reading Segment From Tape With PL/I Calls to iox_ (via tape_mult_)

```
                    call iox_$close (iocbp, (0)); /* Close the switch. */
               end;
               call iox_$detach_iocb (iocbp, (0)); /* Demount the tape */
          end;

          call hcs_$terminate_noname (p, (0)); /* Clean up address space */
     end;
     return;

%include iox_modes;                    /* defines "Stream_input" */
     end;
```

Figure 6-8 (Cont). Reading Segment From Tape With PL/I Calls to iox_ (via tape_mult_)

The tape_in and tape_out commands allow magnetic tape I/O to be performed to and from Multics segments. These commands require the preparation of a control file, in which detailed specification of the I/O operations to be performed can be given.

The following command:

```
tape_out data_tape
```

writes out the segment data, and the following command:

```
tape_in data_tape
```

reads it back in. The file data_tape.tcl, in the working directory, should have the following contents:

```
Volume: 050015;
File: data;
path: >udd>Work>Green>data;
mode: ascii;
format: S;
number: 1;
block: 600;
record: 200;
End;
```

These techniques have the advantage that the simplest command of all can be used to write the segment out or read it back in. Another feature is the standardized control language, which is independent of I/O module, provided by the tape_in and tape_out commands.

The disadvantages of these techniques are the necessity to prepare the control file and maintain it, and the specification of a large amount of detail in it. The control file for these requests specifies ASCII-encoded tape; the ANSI tape standard does not allow for arbitrary or binary data.

INDEX

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 68)
MULTICS PROGRAMMERS' MANUAL
PERIPHERAL INPUT/OUTPUT

ORDER NO. AX49-01

DATED NOVEMBER 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

26134, 7.5C1179, Printed in U.S.A.

AX49-01