To: Distribution

From: Bill Silver

Date: October 10, 1974

Subject: Resource Control Package

INTRODUCTION

This document describes the Resource Control Package (rcp_), a proposed new interface for device and volume management. rcp_ will exist within the security kernel and execute in ring 1. It is strictly an internal interface and will be documented only in PLMs. rcp_ will maintain a list of most of the devices and all of the volumes known to the system. rcp_ will provide the interface to ioi_ (see MTB-056) for the attachment and detachment of devices. It will perform the tape drive and tape reel management functions mentioned in MTB-109.

This document will describe the initial implementation of rcp_. rcp_ should be installed before ioi_ is used as a standard system interface. In order for it to be available as soon as possible, the initial implementation will provide only those device and volume services that are immediately necessary. It is hoped, however, that the interfaces described in this document will be correct and permanent. All the interfaces described in this document will be supported in some manner in the initial implementation. However, in some cases, full support will not be available until later implementations. Such cases will be noted in the document. It is expected that additional capabilities will be provided by rcp_ at a later time. The important consideration in the initial implementation of rcp_ is that all of the I/O modules that will call rcp_ use an interface that will change as little as possible. rcp_ will be made to work, then in the future, the many desirable enhancements to rcp_ can be implemented.

This document contains sections on device management, volume management, calling sequences, a sample scenario, and planned extensions to rcp_. Following this document will be several additional MTBs that will describe parts of rcp_ in more detail and will describe improvements and extensions to rcp_.

DEVICE MANAGEMENT

The initial implementation of rcp_ will perform the following device management functions:

1. assignment and unassignment
2. attachment and detachment
3. special control functions
4. access control
5. maintenance of device information

RCP Data Base

In order to perform its device management functions, rcp_ will maintain a data base containing information about each device that can be under its control. This data base will be set up at system initialization time from the information found on the "prph" configuration cards. The exact information maintained about each device depends upon the generic class of the device. Among other things this information will include:

1. device class
2. device name
3. device model number
4. other information from the prph card
5. state of the device
6. process ID if assigned
7. disposition values

In the initial implementation this data base will be reinitialized each time the system is initialized. It will not be maintained across bootloads. However, since rcp_ will execute in ring 1, it will run on behalf of some process. Thus rcp_ will manage its data base in such a way that it can continue functioning when a process has terminated, even if that process had the rcp_ data base locked and was updating it.

Cold Boot Environment

In the initial implementation all of rcp_ will reside on the Multics system tape. It will be fully operational when Multics leaves ring 0 for the first time. Thus rcp_ can be used to manage tape drives in a cold boot environment. Future implementations of rcp_ may support features that cannot function in a cold boot environment. rcp_ will then be split into two parts. One part will reside on the Multics system tape and will be able to perform basic device management functions in a cold boot environment. The other part will not be on the Multics system tape. It will be initialized when a more suitable process environment is available. The interfaces to rcp_ will be the same regardless of the environment in which it is running.

Device Classes

Each device managed by rcp_ is considered to be a member of a generic device class. The device classes that rcp_ will support in its initial implementation are listed below. Future implementations may support additional device classes. The current device classes are:

tape	(magnetic tape)
printer	(line printer)
punch	(card punch)
reader	(card reader)
console	(operator's console)

Device Names

Each device managed by rcp_ will have a unique device name. In the initial implementation this implies that the device name has been registered with ioam_. This will be done at system initialization time. Any caller that asks rcp_ to assign a specific device (as opposed to asking rcp_ to assign any appropriate device from a device class) will have to know the correct name of the device.

Device names are derived from the names found on the prph configuration cards. For devices that have exclusive use of a channel, such as printers, the device name will be the actual prph configuration card name. For devices that are multiplexed over one or more channels, such as tapes, the device name will have the form, "pppp_xx", where "pppp" is the prph configuration card name and "xx" is a unique identifying number. Such devices will be numbered starting from 1. The maximum number is 63. Some examples of device names are:

rdrb	-	a card reader
tape_02	-	tape drive number 2

Gates

One of the primary functions of rcp_ is to control access to ioi_. Most ioi_ entry points can be called from the user ring via the gate ioi_. Certain entry points in ioi_ will be callable only from the administrative ring (ring 1). A special gate with ring brackets (0,0,1) must be used when calling ioi_ entry points that attach and detach devices or establish limits for the ioi_ workspace size and the ioi_ time-out interval. By permitting these ioi_ entry points to be callable from only the administrative ring, user ring programs are forced to call rcp_.

rcp_ can then make the access checks that will determine whether or not the action requested by the caller may be performed. Device access control will be discussed in one of the following sections.

rcp_ itself may be called via three gates. Each of these three gates will have ring brackets of (1,1,5). The description of each rcp_ entry point will indicate which gate must be used in order to call that entry point. The names and general function of these three gates are:

rcp_	- Most rcp_ entry points can be called through this gate.
rcp_priv_	- This gate will be used by system processes to perform privileged administrative operations.
rcp_tand_	- This gate will be used by special T&D processes. It will allow them to call special privileged rcp_ attachment entry points.

Operator Communication

The administrative gate to ioi_ will be used by rcp_ to access a new privileged ring 0 message program. This privileged ring 0 message will be callable only through this administrative gate. It will be called with a syserr code and a message that will be passed to syserr. However, it will not accept syserr code 1 (crash the system) or syserr code 2 (terminate the process). This new ring 0 message is needed by the initial implementation of rcp_ in order to communicate with the operator.

Device Assignment and Unassignment

Device assignment involves the allocation of a device to a specific process. The process, in a sense, will have temporary ownership of the device for the duration of the assignment. The device will be unavailable to all other processes. Attachment is a separate function which is described below.

Unassigning a device involves taking a device away from a process. If a device is attached when it is being unassigned, rcp_ will first detach the device. When a process terminates, the answering service will make a special call to rcp_ (rcp_$unassign_proc) to force the unassignment of all devices still assigned to the process.

Device Attachment and Detachment

Device attachment involves making a device available for actual I/O processing. For rcp_, this involves calling ioi_ to attach the device. rcp_ will also call ioi_ to initialize all other data needed to use the device. For appropriate device classes, rcp_ will ensure that the correct volume has been mounted and is ready to be used. When the attachment is completed the caller will have all of the information needed to perform I/O on the device.

Detaching a device involves breaking the connection with ioi_ for a device. rcp_ will call ioi_ to detach the device. ioi_ will not accept calls to perform I/O on the device until it is attached again.

The relationship between assignment and attachment is such that assignment does not imply attachment but attachment does imply assignment. Only assigned devices can be attached. If rcp_ is called to make an attachment to a device that is not assigned, it will automatically attempt to assign the device. The inverse is also true, a device may be detached without being unassigned, however, in order to be unassigned it must be detached.

In the initial implementation of ioi_, a call to ioi_ to attach a device will result in ioi_ calling ioam_ to assign the device. Thus initially, the system-wide focal point for the assignment of devices will be ioam_. For some time at least, it will be possible to assign devices of certain classes directly through ioam_. Even if a device belongs to a device class which rcp_ does support, neither rcp_ nor ioi_ will know when a device has been assigned in this way. As far as rcp_ is concerned the device will be available. rcp_ will be able to successfully assign the device. However, ioi_ will not be able to successfully attach the device. This problem will be solved when all of the existing ring 0 device interface modules for device classes supported by rcp_ have been removed. In the initial implementation this problem will not exist for tapes or the operator's console.

Special Device Control Functions

In addition to device assignment and attachment, rcp_ will allow certain privileged processes to add and delete devices from the system. This, of course, can be performed only on the set of devices specified by the prph configuration cards. Only devices that have previously been deleted can be added. A device that has been deleted cannot be assigned by any process.

Access Control

The initial implementation of rcp_ will provide device access control that is essentially the same as that provided by the current system. This means that for each device an "ad hoc" set of rules will be used to determine whether or not a process may assign a device. These rules involve checking the access that a process has to various gate segments. A process will be able to assign a device if it has "E" access to the gate that rcp_ associates with that device. The rules for the various device classes are:

tape - No special access is currently needed to assign a tape drive. rcp_ will enforce an installation defined limit to the number of tape drives that a single process may have assigned at one time. In addition, rcp_ will ensure that an installation defined number of tape drives are reserved for use by system processes. rcp_ considers a system process to be any process that has access to the gate hphcs_. rcp_ will always assign a tape drive to a system process if there is a free drive available. However, rcp_ will not assign a tape drive to a user process if the number of tape drives available is less than the number of tape drives reserved for system processes minus the number of tape drives currently assigned to system processes.

printer - Line printers will be assigned only to processes that have access to the gate prtdcm_.

punch - Card punches will be assigned only to processes that have access to the gate phcs_.

reader - Card readers will be assigned only to process that have access to the gate phcs_.

console - The operator's console will be assigned only to processes that have access to the gate rcp_tand_.

Device Information

rcp_ will contain entry points that may be called to obtain information about devices that are under its control. They will not be available in the initial implementation. These entry points are similiar to the entry points provided by ioam_. However, the corresponding entry points in ioam_ will not be deleted. ioam_ will still be the only system interface which can return information about all of the devices that are assigned to a process. In the initial implementation of rcp_ and ioi_, ioam_ will still be called to assign each device that is assigned by rcp_. The opposite is not true, however, since some devices may be assigned through ioam_ which are not assigned through rcp_. For the initial implementation no changes will have to be made to any program that calls ioam_.

VOLUME MANAGEMENT

The only volumes which rcp_ will manage in its initial implementation are tape reels. A comprehensive tape reel management capability is planned as one of the first major enhancements to rcp_. The initial implementation, however, will perform only limited tape reel management. A detailed description of the tape reel management functions that are performed by the initial implementation of rcp_ will be given in the description of the calling sequence for the rcp_$attach_tape entry point.

ENTRY POINTS

This section contains a list of the entry points in rcp_. The description of each entry point includes the arguments that it accepts, a discussion of the function of the entry point, and any necessary notes about the entry point. The segment name associated with each entry point is the name of the gate that must be called in order to access that entry point. The standard error_table_ codes that may be returned by each entry point have not been listed. The caller should consider any non zero error code returned by any rcp_ entry point as an indication of a fatal error.

Device Assignment and Attachment Entry Points

```
rcp_$assign_tape (event_id,   comment,   tape_info_ptr,   rcp_id,
                  error_code)
```

ARGUMENTS:

event_id (Input) (fixed bin(71)) This is the ipc_ event channel ID that will be used to check this assignment. It will be used to send wakeups that signal the possible completion of the assignment.

comment (Input) (char(*)) This string is a comment that will be displayed to the operator. It will be displayed after rcp_ has successfully completed the assignment.

tape_info_ptr (Input) (ptr) A pointer to a structure provided by the caller. This structure contains information about the tape drive that is to be assigned. This structure is defined below. rcp_ considers each field in this structure to be an input argument to rcp_$assign_tape.

```
dcl 1 tape_info  based (tape_info_ptr),
      2 version_num  fixed bin,   /* 1. */
      2 tape_drive   char(8),     /* 2. */
      2 tracks       fixed bin;   /* 3. */
```

1. version_num - This field must be set by the caller. It tells rcp_ what version of this structure the caller is programmed to use. In the initial implementation rcp_ will expect it to be set to 1.

2. tape_drive - This field should contain either "tape" or the device name of the specific tape drive that the caller wants to assign. If it contains "tape" rcp_ will assign any appropriate tape drive that is available.

3. tracks - This field represents the requested track type of the tape drive that is to be assigned. It is used only when tape_info.tape_drive contains "tape". The acceptable values are:

```
9 => assign a 9 track drive
7 => assign a 7 track drive
0 => use default track type
```

If this field equals 0 a 9 track drive will be assigned by default

rcp_id (Output) (bit(36) aligned) This is rcp_'s unique identifier for this assignment. It is valid until this assignment is terminated.

error_code (Output) (fixed bin(35)) This is a standard error_table_ code.

FUNCTION:

rcp_$assign_tape initiates the assignment of a tape drive. The tape drive that is assigned is dependent upon the values in tape_info.tape_drive and tape_info.tracks. If tape_info.tape_drive does not contain "tape" then rcp_ will assume that it contains the device name of the tape drive to be assigned. If this device name does not specify a known tape drive, or if the specified tape drive cannot be assigned, rcp_ will abort the assignment. rcp_ will make no attempt to assign any other tape drive. If tape_info.tape_drive does contain "tape" then rcp_ will assign any available and appropriate tape drive. In this case, rcp_ will ensure that the tape drive assigned has the track type specified in tape_info.tracks.

In either case, rcp_ will consider the access that the process has to nphcs_ in order to determine whether or not this is a system process. If it is, rcp_ will increment the count of tape drives assigned to system processes. If it is not, rcp_ will ensure that the limit placed on the number of tape drives that may be concurrently assigned by all user processes is not exceeded. rcp_ will also enforce the limit placed on the number of tape drives that a single process may have assigned at one time.

Each device assigned has associated with it a disposition value. This disposition value tells rcp_ whether or not the device should be unassigned when it is detached. When a device is assigned by an rcp_ assignment entry point its disposition value will be initialized to specify that the device is not to be unassigned if it is detached. This disposition value may be overridden by subsequent calls to rcp_. (See rcp_$detach.) However, in most cases explicitly assigning a device via a call to an rcp_ assignment entry point implies that the device must be unassigned by a call to rcp_$unassign.

This entry point functions in cooperation with the rcp_$check_assign entry point. rcp_$assign_tape only initiates the assignment of a tape drive. When it returns, the drive will not yet be assigned to the caller's process. A call must be made to rcp_$check_assign in order to obtain the drive name and track type of the tape drive that will be assigned. Any attempt to attach this device before the assignment is complete will result in an error. (See rcp_$check_assign.)

NOTES:

All of the rcp_ assignment and attach entry points function in cooperation with a corresponding rcp_ check entry point. The need for the check entry points may not be immediately obvious. They are needed because, in some cases and in future implementations, the assignment or attachment functions performed by rcp_ cannot be completed without blocking and waiting for one or more events to occur. The check entry points force the caller to use a programming sequence that provides for repeated blocking. (See the section containing the sample scenario of calls to rcp_.) The user must do the blocking and not rcp_ since rcp_ will execute in a lower ring and it is undesirable to block in a lower ring. The events that will be waited for depend upon the class of the device involved, the current state of the devices and volumes, and the implementation of rcp_. For example, the attachment of tape drives involves the mounting of a tape reel and this involves I/O operations and they involve waiting for wakeups from ioi_. In future implementations of rcp_ all device assignment functions may be performed by some system process and this involves waiting for that process to send a wakeup that signals the completion of the assignment.

rcp_$attach_tape (event_id, comment, tape_info_ptr, reel_name, write_flag, reg_data_ptr, rcp_id, error_code)

ARGUMENTS:

event_id (Input) (fixed bin(71)) This is the ipc_ event channel ID that will be used to check this attachment. It will be used to send wakeups that signal the possible completion of the attachment. This event channel ID is passed to ioi_ when ioi_ is called to attach the device. rcp_ treats this event channel ID independently from the event channel ID specified in any previous assignment call.

comment (Input) (char(*)) This string is a comment that will be displayed to the operator. It will be displayed after rcp_ has successfully completed the attachment.

tape_info_ptr (Input) (ptr) (See rcp_$assign_tape.) One minor difference exists between the way rcp_$attach_tape and rcp_$assign_tape use the tape_info structure. If tape_info.tracks equals 0 then rcp_$attach_tape, in later implementations, will look in the tape reel registration data for the track type of the tape drive to assign. Only if it is unspecified in the registration data will the default of 9 tracks be used.

reel_name (Input) (char(*)) This string contains the name of the tape reel that the caller wants mounted on the attached tape drive.

write_flag (Input) (bit(1)) If this flag is ON rcp_ will ensure that the specified tape reel is mounted with a write ring. If it is OFF rcp_ will ensure that the specified tape reel is mounted without a write ring.

reg_data_ptr (Input) (ptr) A pointer to a structure provided by the caller. This structure defines the tape reel registration data that is returned by rcp_. (See the notes below.)

rcp_id (Output) (bit(36) aligned) This is a unique identifier that rcp_ generates when a device is assigned. rcp_ does not generate a new rcp_id when the device is attached. It is used to identify the assigned device that is being attached.

error_code (Output) (fixed bin(35))

FUNCTION:

rcp_$attach_tape will initiate the attachment of a tape drive. The tape drive that is attached is dependent upon the values in tape_info.tape_drive and tape_info.tracks. The basic strategy for determining which tape drive to attach is the same as that used by rcp_$assign_tape. rcp_ will check to see if an appropriate tape drive is currently unattached but assigned to this process. If rcp_ finds an appropriate tape drive that is already assigned it will initiate the attachment of that tape drive. If rcp_ cannot find an appropriate tape drive that is already assigned it will initiate the assignment of a tape drive as previously described. If rcp_ has a choice among several appropriate tape drives and one of them already has mounted the tape reel that is needed for this attachment then rcp_ will assign that tape drive.

Once rcp_ has found an acceptable and assigned tape drive it will initiate the attachment of the device. rcp_ will call ioi_ to perform the ring 0 device attachment. If this is successful, rcp_ will initiate the mounting of the specified tape reel. A tape reel that is mounted or is being mounted is considered to be assigned to the process that the associated tape drive is assigned to. If the specified tape reel is already assigned then rcp_ will not be able to complete the assignment at this time. (See rcp_$check_attach.)

rcp_ will recognize the case where this tape reel is already mounted and will not issue a redundant request to the operator to have it mounted.

There are two subtle differences between the case where a device has been explicitly assigned by a call to an rcp_ assignment entry point and the case where a device has been automatically assigned by an rcp_ attachment entry point. These differences are:

1. Assume that the device has been assigned successfully. Now assume that while attaching the device a fatal error occurs. The attachment will be abandoned. If the device was assigned by the attachment entry point then the device will be unassigned. However, if the device was previously assigned by an assignment entry point then it will remain assigned.

2. If the device being attached was previously assigned by an assignment entry point then the disposition value associated with this device will not be changed. If the device has been assigned by the attachment entry point then the disposition value for this device will be initialized to specify that the device is to be unassigned when it is detached. (See rcp_$detach.)

This entry point functions in cooperation with the rcp_$check_attach entry point. rcp_$attach_tape initiates the attachment but does not complete it. The caller still can not successfully call ioi_ to perform I/O on the tape drive being attached. The attachment will not be completed and the caller will not know the device name of the tape drive that was actually attached until a successful call is made to rcp_$check_attach.

NOTES:

The reg_data_ptr argument should point to a structure provided by the caller. This structure will contain the tape reel registration data that is returned by rcp_. rcp_ will return all of the tape reel registration data that is needed by the tape I/O module as well as all of the tape reel registration data that may be changed by a user process. This information is returned by this entry point for the convenience of the calling tape I/O module.

The initial implementation of rcp_ will not support tape reel registration data. However, the fields in this structure will be filled in by the initial version of rcp_. Each field will be set to a default value that will be valid even when the full tape reel management facility is available. These default values will specify that the information associated with the field is not known by rcp_. The registration data structure is as follows:

```
dcl 1 reg_data           based (reg_data_ptr),
      2 version_num      fixed bin,      /*  1. */
      2 volume_id        char(32),       /*  2. */
      2 tracks           fixed bin,      /*  3. */
      2 density          fixed bin,      /*  4. */
      2 label_type       fixed bin,      /*  5. */
      2 usage_count      fixed bin,      /*  6. */
      2 read_errors      fixed bin,      /*  7. */
      2 write_errors     fixed bin;      /*  8. */
```

1. version_num - (See tape_info.version_num.)

2. volume_id - This field will contain the volume ID of the tape reel. In the initial implementation rcp_ will set it to the contents of the reel_name argument.

3. tracks - This field represents the track type specified in the registration data for this tape reel. In the initial implementation rcp_ will set it to 0.

4. density - This field represents the tape density specified in the registration data for this tape reel. In the initial implementation rcp_ will set it to 0.

5. label_type - This field represents the label type specified in the registration data for this tape reel. In the initial implementation rcp_ will set it to 0.

6. usage_count - This field is a count of the total number of times this tape reel has been attached by any process since the last time the field was reset. The current attachment will be included in this count. In the initial implementation rcp_ will set it to 1.

7. read_errors - This field is a count of the total number of read errors that have occurred with this tape reel since the last time this field was reset. In the initial implementation rcp_ will set it to 0.

8. write_errors - This field is a count of the total number of write errors that have occurred with this tape reel since the last time this field was reset. In the initial implementation rcp_ will set it to 0.

rcp_tand_$priv_attach_tape (event_id, comment, tape_info_ptr, reel_name, write_flag, reg_data_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$attach_tape.

FUNCTION:

rcp_tand_$priv_attach_tape functions basically the same as rcp_$attach_tape. The only difference is that when rcp_ calls ioi_ to perform the ring 0 attachment of the tape drive it will call a privileged ioi_ attach entry point. This privilege is given by ioi_. With this privilege ioi_ will allow subsequent connect calls to specify a PCW as well as a list of DCWs, and to address device 0.

rcp_$assign_printer (event_id, comment, printer_info_ptr, rcp_id, error_code)

ARGUMENTS:

event_id (Input) (fixed bin(71))
(See rcp_$assign_tape.)

comment (Input) (char(*)) (See rcp_$assign_tape.)

printer_info_ptr (Input) (ptr) A pointer to a structure provided by the caller. This structure is similar in format and function to the tape_info structure.

```
dcl 1 printer_info    based (printer_info_ptr),
      2 version_num   fixed bin,  /* 1. */
      2 printer       char(8),    /* 2. */
      2 model         fixed bin,  /* 3. */
      2 chain         fixed bin;  /* 4. */
```

1. version_num - (See tape_info.version_num.)

2. printer - This field should contain either "printer" or the device name of the specific printer that the caller wants to assign. If it contains "printer" then rcp_ will assign any appropriate printer that is available.

3. model - This field represents the requested model number of the printer that is to be assigned. It is used only when printer_info.printer contains "printer". It must be a value that is found in the "model" field of a printer prph configuration card. If it is 0 rcp_ will not consider the model characteristic in its search for a printer to assign.

4. chain - This field represents the requested chain (print train) type of the printer that is to be assigned. It is used only when printer_info.printer contains "printer". It must be a value found in the "chain" field of a printer prph card. If it is 0 rcp_ will not consider the chain characteristic in its search for a printer to assign.

rcp_id (Output) (bit(36) aligned)
(See rcp_$assign_tape.)

error_code (Output) (fixed bin(35))

FUNCTION:

rcp_$assign_printer will initiate the assignment of a printer. Before initiating the assignment, rcp_ will verify that the calling process has the proper access. The device assigned is dependent upon the fields in printer_info. If printer_info.printer does not contain "printer" then rcp_ will assume that it contains the device name of the printer to be assigned. If this device name is invalid, or if this device cannot be assigned, rcp_ will abort the assignment. If printer_info.printer does contain "printer" then rcp_ will assign any available and appropriate printer. rcp_ will ensure that the printer assigned has the model and chain characteristics specified in printer_info.model and printer_info.chain. If either of these arguments is 0 rcp_ will not consider that characteristic in its search for an appropriate device. rcp_$assign_printer functions and cooperates with rcp_$check_assign in the same manner that was described for rcp_$assign_tape.

rcp_$attach_printer (event_id, comment, printer_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$assign_printer.

FUNCTION:

rcp_$attach_printer will initiate the attachment of a printer. The printer that is attached is dependent upon the fields in printer_info. The basic strategy for determining which printer to attach is the same as that used by rcp_$assign_printer. If rcp_ cannot find an appropriate printer that is already assigned it will initiate the assignment of a printer as previously described. Once rcp_ has found an acceptable and assigned printer it will initiate the attachment of that printer. rcp_ will call ioi_ to perform the ring 0 device attachment. rcp_$attach_printer functions and cooperates with rcp_$check_attach in the same manner that was described for rcp_$attach_tape.

rcp_tand_$priv_attach_printer (event_id, comment, printer_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$assign_printer.

FUNCTION:

rcp_tand_$priv_attach_printer has the same relationship to rcp_$attach_printer that rcp_tand_$priv_attach_tape has to rcp_$attach_tape.

rcp_$assign_punch (event_id, comment, punch_info_ptr, rcp_id, error_code)

ARGUMENTS:

The arguments to this entry point are similar to the arguments to rcp_$assign_printer. The only differences are that they reference punch devices rather than printer devices and that there is no chain field in the punch info structure.

FUNCTION:

This entry point functions basically the same as the rcp_$assign_printer entry point.

rcp_$attach_punch (event_id, comment, punch_info_ptr, rcp_id, error_code)

ARGUMENTS

This entry point has the same arguments as the rcp_$assign_punch entry point.

FUNCTION:

This entry point functions basically the same as the rcp_$attach_printer entry point.

rcp_tand_$priv_attach_punch (event_id, comment, punch_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$assign_punch.

FUNCTION:

rcp_tand_$priv_attach_punch has the same relationship to rcp_$attach_punch that rcp_tand_$priv_attach_tape has to rcp_$attach_tape.

rcp_$assign_reader (event_id, comment, reader_info_ptr, rcp_id, error_code)

ARGUMENTS:

The arguments to this entry point are similar to the arguments to rcp_$assign_printer. The only differences are that they reference card reader devices rather than printer devices and that there is no chain field in the reader_info structure.

FUNCTION:

This entry point functions basically the same as the rcp_$assign_printer entry point.

rcp_$attach_reader (event_id, comment, reader_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as the rcp_$assign_reader entry point.

FUNCTION:

This entry point functions basically the same as the rcp_$attach_printer entry point.

rcp_tand_$priv_attach_reader (event_id, comment, reader_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$attach_reader.

FUNCTION:

rcp_tand_$priv_attach_reader has the same relationship to rcp_$attach_reader that rcp_tand_$priv_attach_tape has to rcp_$attach_tape.

rcp_$assign_console (event_id, comment, console_info_ptr, rcp_id, error_code)

ARGUMENTS:

event_id (Input) (fixed bin(35))
(See rcp_$assign_tape.)

comment (Input) (char(*)) (See rcp_$assign_tape.)

console_info_ptr (Input) (ptr) A pointer to a structure provided by the caller. Since current Multics systems have only one operator's console, this structure will not really be used when assigning or attaching the operator's console. It is included in this calling sequence for the sake of consistency and because it may be needed by future implementations of rcp_.

```
dcl 1 console_info   based (console_info_ptr),
      2 version_num  fixed bin,  /* 1. */
      2 console      char(8),    /* 2. */
      2 model        fixed bin;  /* 3. */
```

1. version_num - (See tape_info.version_num.)

2. console - This field should contain "console".

3. model - This field is not currently used for assignment or attachment. When the console_info structure is filled in by rcp_$check_assign or rcp_$check_attach this field will contain one of the following values.

   1 => "ibm" - BCD console
   2 => "emc" - Entry Mode Console
   3 => "scc" - System Control Center

rcp_id (Output) (bit(36) aligned)
(See rcp_$assign_tape.)

error_code (Output) (fixed bin(35))

FUNCTION:

Since there is only one operator's console available on the current system, rcp_ will not have to search for an appropriate console device to assign. If the calling process has access to the gate rcp_tand_, and if the console is not already assigned, then rcp_ will assign the console to the calling process. A call must must be made to rcp_$check_assign in order to complete the assignment.

rcp_$attach_console (event_id, comment, console_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$assign_console.

FUNCTION:

rcp_$attach_console initiates the attachment of the operator's console. If the operator's console is not already assigned to the calling process then rcp_ will attempt to assign it. Once the console has been successfully assigned to this process rcp_ will initiate the attachment of the console. Before rcp_ can call ioi_ to attach this device, rcp_ must call the ring 0 console device interface module, ocdcm_, to detach the console from the syserr mechanism. If the syserr recovery mechanism is available it will be enabled by ocdcm_. rcp_ will then call ioi_ to perform the ring 0 attachment. In order to complete the attachment the caller must call rcp_$check_attach.

rcp_tand_$priv_attach_console (event_id, comment,
console_info_ptr, rcp_id, error_code)

ARGUMENTS:

This entry point has the same arguments as rcp_$assign_console.

FUNCTION:

rcp_tand_$priv_attach_console has the same relationship to rcp_$attach_console that rcp_tand_$priv_attach_tape has to rcp_$attach_tape.

rcp_$check_assign (rcp_id, device_info_ptr, state_index,
error_code)

ARGUMENTS:

rcp_id (Input) (bit(36) aligned) This argument identifies the assignment request to be checked.

device_info_ptr (Input) (ptr) A pointer to a structure provided by the caller. The format of this structure depends upon the class of the device whose assignment is being checked. It should correspond to the structure referenced by the assignment call. It may or may not be in the same location as the structure referenced by the assignment call. All the fields in this structure except the version number will be used as output arguments by rcp_. rcp_ will use these output fields to return the device name and other characteristics of the device that was actually assigned.

state_index (Output) (fixed bin) This argument represents the state of the assignment. More detailed information about this argument is given below. The values that may be returned are:

0	=>	ready
1	=>	short wait
2	=>	long wait
3	=>	fatal error

error_code (Output) (fixed bin(35)) This value will be 0 unless the value of the state_index is 3. In this case, an error_table_ code will be returned.

FUNCTION:

rcp_$check_assign functions in cooperation with the rcp_ assignment entry points. After calling one of the assignment entry points another call must be made to this entry point. rcp_$check_assign will see if the assignment has been completed.

If the assignment is proceeding normally but has not yet completed, the caller will be told that there is a short wait. A state_index value of 1 will be returned. The situations that may result in a short wait condition vary for each device class. They will also vary with future implementations of rcp_. An example of a short wait situation would be: rcp_ is waiting for the operator to give permission to assign a printer to the calling process. This is a case that may actually happen with future implementations of rcp_. The caller will not be told the reason for the short wait. What the caller does know is that, in order to signal the possible end of the short wait condition, a wakeup will be sent over the ipc_ event channel whose ID was passed as an argument in the assignment call that is being checked. The procedure that sends this wakeup depends upon the situation and the implementation of rcp_. It may be rcp_ itself, or it may be ioi_, or it may be a system process that is involved in the device assignment. The caller should block on this event channel whenever a state_index of 1 is returned. When the wakeup comes through the caller should not assume that the assignment has been completed. He may correctly assume only that it might have been completed. He must call rcp_$check_assign again. This whole sequence must be repeated until rcp_$check_assign returns a state_index value of 0 or indicates an error condition.

If the assignment cannot be completed because the device (and possibly the volume) to be assigned is already assigned to another process, rcp_ will return a state_index value of 2. rcp_ can make no predictions about when the needed resource will become available. rcp_ does know that some time in the future the resource will be available and the assignment can be completed. This is the long wait case. If the caller chooses to wait he should block just as with the short wait case. Otherwise, he should call rcp_$unassign to abandon the assignment. In the initial implementation of rcp_ the long wait case is not supported. Instead, the caller will be told that the assignment cannot be made.

When the assignment is completed rcp_ will return the device name and associated characteristics of the assigned device. They will be returned in the device_info structure. The information returned by rcp_$check_assign will be valid only when a state_index value of 0 is returned.

```
rcp_$check_attach (rcp_id,      device_info_ptr,       ioi_index,
                   workspace_max,   timeout_max,     state_index,
                   error_code)
```

ARGUMENTS:

rcp_id (Input) (bit(36) aligned) This argument identifies the attachment request to be checked.

device_info_ptr (Input) (ptr) (See rcp_$check_assign.)

ioi_index (Output) (fixed bin) This is the device index generated by ioi_. It must be used in all subsequent calls to ioi_ for this device during this attachment.

workspace_max (Output) (fixed bin) This is the maximum size of the ioi_ workspace for the assigned device. ioi_ will reject any attempt to expand the workspace beyond this limit.

timeout_max (Output) (fixed bin) This is the maximum time-out interval that ioi_ will allow. ioi_ will reject any attempt to set a time-out interval that is greater than this limit.

state_index (Output) (fixed bin)
(See rcp_$check_assign.)

error_code (Output) (fixed bin(35))

FUNCTION:

rcp_$check_attach functions in cooperation with the rcp_ attachment entry points. After calling one of the attachment entry points another call must be made to this entry point. rcp_$check_attach will see if the attachment has been completed. The attachment of a tape drive is not complete until the requested tape reel is mounted and is ready for I/O processing.

If the attachment has completed successfully, rcp_ will then perform the final steps necessary for the caller to perform I/O on the attached device. rcp_ will call ioi_ to set the maximum workspace size and the maximum time-out interval. It is rcp_ that defines these limits. ioi_ enforces them. (See Appendix A for a list of these limits for each device class.) rcp_ will call ioi_ to promote the validation level for this device to the caller's validation level. rcp_$check_attach will return the device name and associated characteristics of the device that was actually attached. This information will be returned in the device_info structure. It will also return the ioi_ device index and the ioi_ limits that have been defined. It will return a state_index value of 0. The output arguments returned by rcp_$check_attach will be valid only when the state_index value returned is 0.

After rcp_$check_attach has indicated that the attachment has been completed, the caller should call ioi_ to set up his I/O environment. The event channel ID that was passed to rcp_ in the attachment call was in turn passed to ioi_. If the caller wants to use a different event channel he may now call ioi_ to change it. rcp_ set up only the limits of the workspace size and the time-out interval. The caller <u>must</u> call ioi_ to establish his workspace. rcp_ does not return a pointer to the ioi_ workspace. rcp_ will leave the time-out interval set to the default value defined by ioi_. Unless the caller wants to use this default value he should call ioi_ to set the time-out interval that he wants.

If the attachment is proceeding normally but has not yet completed, the caller will be told that there is a short wait. A state_index value of 1 will be returned. The situations that may result in a short wait vary for each device class. They will also vary with future implementations of rcp_. An example of a short wait situation would be: rcp_ is waiting for a tape reel to be mounted and to become ready. The short and long wait cases that were discussed with rcp_$check_assign are also possible with this entry point since the attachment that is being checked may also have involved an assignment. rcp_$check_attach will not return a state_index value of 0 until both the assignment and the attachment have been completed. (See rcp_$check_assign.)

rcp_$detach (rcp_id, comment, disposition, error_code)

ARGUMENTS:

rcp_id (Input) (bit(36) aligned) This argument identifies the currently attached device that is to be detached.

comment (Input) (char(*)) This string is a comment that will be dispdayed to the operator after the device has been detached. This argument allows the caller to give some final instructions to the operator regarding this device.

disposition (Input) (fixed bin) This argument specifies the action to be taken by rcp_ with regard to the disposition of the device being detached. The disposition of the device involves the possible retention of the device assignment even though the device is being detached. The legal values which this argument may have are:

0 => unspecified
1 => unassign the device
2 => retain the device assignment

error_code (Output) (fixed bin(35))

FUNCTION:

rcp_$detach will always detach the specified device. Detaching implies that rcp_ will call ioi_ to detach this device in ring 0. The ioi_ index that had been associated with this device is now invalid. ioi_ will reject all calls for this device until it is attached again. In the initial implementation rcp_$detach will always unassign any volume mounted on the device being detached This implies that this volume may be assigned to another process. rcp_ may or may not actually dismount this volume depending upon what it considers to be the best course of action at the time.

If the disposition argument contains 0 then rcp_ will use the current disposition value associated with the device in order to determine whether or not it should unassign the device. If the device was assigned by an attachment entry point then it will be unassigned. If the device was assigned by an assignment entry point it will not be unassigned. I/O modules should always call rcp_$detach with a disposition argument of 0 unless they have some good reason to explicitly specify the disposition.

If the disposition argument contains 1 then rcp_$detach will always unassign the device. The rcp_id associated with this device will no longer be valid. The device will be available for assignment to other processes. If the disposition argument contains 2 then rcp_$detach will not unassign the device. The device will still be assigned to the caller's process. The rcp_id associated with this device assignment will still be valid. The device will be available for future attachments by this process.

rcp_$unassign (rcp_id, comment, error_code)

ARGUMENTS:

rcp_id (Input) (bit(36) aligned) This argument identifies the device that is to be unassigned.

comment (Input) (char(*)) (See rcp_$detach.)

error_code (Input) (fixed bin(35))

FUNCTION:

This entry point will unassign the specified device. Any previous disposition specification will be ignored. If the device is currently attached rcp_ will automatically detach it before it is unassigned. (See rcp_$detach.)

rcp_$update_tape_reg (rcp_id, reg_data_ptr, error_code)

ARGUMENTS:

rcp_id (Input) (bit(36) aligned) This argument identifies the attached tape drive on which the tape reel involved with this update is mounted.

reg_data_ptr (Input) (ptr) A pointer to a structure provided by the caller. This structure defines the tape reel registration data that will be updated. See the discussion of rcp_$attach_tape for a description of this structure.

error_code (Output) (fixed bin(35))

FUNCTION:

rcp_$update_tape_reg will update the tape reel registration data associated with the tape reel that is mounted on the specified tape drive. This entry point will exist in the initial implementation of rcp_ but it will not perform any update function. Even so, tape I/O modules that use the initial implementation of rcp_ should call this entry point when it is appropriate to do so. Later documentation dealing with rcp_ will describe which fields in the registration data structure are updated and the exact manner in which the update is made for each field.

Administrative Entry Points

rcp_priv_$force_unassign (device_name, error_code)

ARGUMENTS:

device_name (Input) (char(*)) This argument specifies the name of the device that rcp_ is to unassign. It must be a legal device name that is known to rcp_.

error_code (Output) (fixed bin(35))

FUNCTION:

This entry point is called to force the unassignment of a specific device. This device does not have to be assigned to the calling process. It is expected that only certain privileged system processes will have access to the gate through which this call must be made. If the device is currently attached, rcp_ will detach the device before it is unassigned. In order to detach a device that is attached to another process, rcp_ will call a privileged entry in ioi_ to force detach the device. ioi_ will then reject any future calls for this device from the process that previously had the device assigned.

rcp_priv_$proc_unassign (process_id, error_code)

ARGUMENTS:

process_id (Input) (bit(36) aligned) This argument contains the process ID of a process that will have all of its assigned devices unassigned.

error_code (Output) (fixed bin(35))

FUNCTION:

This entry point is called to unassign all of the devices rcp_ has assigned to the specified process. See the function description of rcp_$force_unassign for details about unassigning a device that is assigned to another process. Normally the calling process will not be the process that is having all of its devices unassigned. This entry point will be called by the initializer process (under the semblance of the answering service) whenever a process terminates. This is done in order to be sure that no devices remain assigned to terminated processes.

rcp_priv_$delete_device (device_name, error_code)

ARGUMENTS:

device_name (Input) (char(*)) This argument specifies the name of the device that rcp_ is to delete.

error_code (Output) (fixed bin(35))

FUNCTION:

This entry point will delete the specified device from the list of devices that rcp_ may assign. The effect is that this device is no longer configured on the system. If the device to be deleted is currently assigned to a process, the device will not be deleted until the device becomes unassigned.

rcp_priv_$add_device (device_name, error_code)

ARGUMENTS:

device_name (Input) (char(*)) This argument specifies the name of the device that rcp_ is to add.

error_code (Output) (fixed bin(35))

FUNCTION:

This entry point will add the specified device to the list of devices which may be assigned by rcp_. The device may then be assigned to any acceptable process that attempts to assign it. The device being added must have been deleted by a previous call to rcp_priv_$delete_device.

SAMPLE SCENARIO

Below is a sample scenario of calls to rcp_. This example represents an acceptable sequence of calls to rcp_ for the purpose of attaching any model 301 printer. This example shows the relationship between the attach entry points and the check_attach entry point. The sequence of calls used to perform just an assignment would be very similar.

```
ATTACH:                                 /* Begin attachment. */
     [set up event channel "ev_id" and wait list.]
     pi_ptr = addr(printer_info);
     pi_ptr->printer_info.version_num = 1;
     pi_ptr->printer_info.printer = "printer";
     pi_ptr->printer_info.model = 301;
     pi_ptr->printer_info.chain = 0;
     call rcp_$attach_printer (ev_id,"Example",pi_ptr,
                                   rcp_id,code);
     if   code ^= 0  then goto ERROR;
CHECK_LOOP:
     call rcp_$check_attach (rcp_id,pi_ptr,ix,wm,tm,sx,code);
     goto STATE(sx);
STATE(1):                               /* Short wait. */
     call ipc_$block (wl_ptr,m_ptr, code);
     if   code ^= 0  then goto ERROR;
     goto CHECK_LOOP;
STATE(2):                               /* Long wait. */
     [Ask user if he wants to wait.]
     [If yes, handle long wait case.]
STATE(3):                               /* Fatal error. */
ERROR:
     [Process error.]
     return;
STATE(0):                               /* Attachment complete. */
     [Get info on attached printer from printer_info.]
     [call ioi_ to  set up I/O environment.]
     [Perform I/O on device.]
```

PLANNED EXTENSIONS

It is hoped that within the framework of the interfaces described in this document many significant improvements can and will be made to rcp_. Below is a partial list of some of these enhancements. They are in no special order.

1. Support for the long wait case of rcp_$check_assign and rcp_$check_attach. This involves solving all of the deadlocking problems that can occur with competing processes.

2. Implementation of an entry point that will perform the assignment of two or more devices.

3. Implementation of an entry point that will perform the assignment of one or more volumes. Corresponding to this will be support for a disposition value that specifies that both the device and volume are to remain assigned and support for a disposition value that specifies that the device is to be unassigned but the volume is to remain assigned.

4. Implementation of an entry point that will set the disposition of a device.

5. Implementation of a tape mount entry point. This will allow the functions of attachment and mounting to be separated. This also implies that detachment and dismounting may be separated.

6. Implementation of device and volume reservation. Reservation is different from assignment. Reservation involves defining a time period, possibly in the future, in which a resource may be assigned to only the specified user. The resource will be unavailable to all other users during that time period.

7. Full implementation of tape reel management. This will include registration of tape reel data, access control lists for each tape reel, operator authentication of tape mounts, label checking, etc.

8. Improved access control over the assignment of devices. This might include access control lists for each device. It might include bringing the assignment of devices under the control of the access isolation mechanism by associating an access level with each device. It might also include operator authorization of device assignments and attachments.

9. Implementation of accounting for device and volume reservations, assignments, and attachments.

10. Implementation of points 6, 7, 8, and 9 above may result in the initializer process or some other system process being used to perform many of the functions of rcp_.

11. Development of the capability to set limits on the length of time that a process may have a device or volume assigned. This time limit could vary depending upon the resource involved and the user involved.

12. Development of the capability to set different ioi_ limits for different users.

13. Development of a complete set of device list and device status entry points.

14. The partial, or possibly complete, replacement of ioam_.

15. Support for disks as a new device class.

16. Implementation of commands that will reserve, assign, and mount devices and volumes.

Appendix A
ioi_ Limits

<u>Device Class</u>	<u>Max Workspace</u>	<u>Max Time-Out</u>
tape	3K	5 minutes
printer	1K	30 seconds
punch	1K	30 seconds
reader	1K	30 seconds
console	1K	3 minutes