MULTICS HASP SERVICE AND UTILITY MANUAL

SUBJECT

Description of Multics HASP Workstation Facility

SPECIAL INSTRUCTIONS

Users of this document should be familiar with certain Multics tables and master files. Refer to the Preface for a list of related documentation.

SOFTWARE SUPPORTED

Multics Software Release 10.2

ORDER NUMBER

GB60-00

October 1983

Honeywell

PREFACE

This manual is intended for system administrators, site support personnel, and system operators. It contains information necessary for setting up a HASP workstation, using the Multics system as the host or as a workstation simulator.

Section 1 is a brief introduction. Sections 2 and 3 contain configuration information for communications and input/output, respectively.

Sections 4 and 5 deal with operation of the workstation; Section 5 contains the daemon driver command descriptions.

Section 6 documents the hasp_host_ and hasp_workstation_ I/O modules.

Section 7 describes testing procedures.

Section 8 provides a short "checklist" in which the main points for setting up your HASP facility are repeated.

File No.: 1L53, 1U53 GB60-00

Setting up HASP requires access to privileged system files and tables. You may need to refer to the I/O daemon tables and daemon driver information in the *Multics Bulk I/O*, manual Order No. CC34. The *Multics Administrator's Manual (MAM) Communications*, Order No. CC75, contains helpful information on the channel master file. You should be familiar with the information in the *Multics Programmer's Reference*, Order No. AG91, especially concerning the terminal type table. Finally, some routine procedures mentioned are described fully in either the *MAM-System*, Order No. AK50, or the *MAM-Project*, Order No. AK51.

CONTENTS

SECTION 1

INTRODUCTION

The HASP (Houston Automatic Spooling Processor) communications protocol is used primarily for remote-job-entry (RJE) tasks. The Multics system can be configured to operate as a host system, communicating with a remote workstation. The remote workstation is usually an RJE terminal with an operator's console and one or more card readers, line printers, and card punches. Alternatively, the Multics system can be configured to simulate a workstation, communicating with a remote host system. Either way, Multics users can request that job decks be transmitted to the host system for execution and returned to the workstation for printing/punching or online perusal.

The HASP communications protocol is supported by the demultiplexing features of the Multics Communication System. This mechanism allows each device of the remote workstation, or each simulated device when Multics is the workstation, to be controlled independently by separate processes. HASP channels, like all communications channels, are defined in the Channel Master File (CMF). The Multics host or workstation requires an entry in the CMF configuring a HASP multiplexer channel. Each device of the workstation, whether Multics is the host or the workstation, also requires an entry defining its channel (device channel) as a slave channel, a separate subchannel of the HASP multiplexer channel.

A process using a device channel attaches that channel through either the hasp_workstation_ I/O module (when Multics is the host) or the hasp_host_ I/O module (when Multics is the workstation). These I/O modules are described in Section 6.

In order to use HASP and actually submit jobs from the workstation, you must first make several decisions and perform the tasks necessary to support them. The bulk of the work lies in modifying the CMF and I/O daemon table. The following sections describe the necessary steps required in setting HASP up for use. Much of the groundwork is the same, regardless of whether the Multics system is acting as host or as workstation. Differences are described when the setup diverges.

SECTION 2

ADMINISTRATIVE SETUP - COMMUNICATIONS

THE FNP CORE IMAGE

The Front-End Network Processor (FNP) software manages the physical communications channels and the synchronous HASP protocol. Thus, support of the HASP communications protocol requires that the proper software be loaded into the FNP supporting the remote workstation terminal or remote host system. The first step is to bind the two modules, bsc_tables and hasp_tables, into the core image before the FNP is loaded. The bsc_tables module takes control of the HASP multiplexer channel when the FNP is told to listen to it. As the HASP multiplexer is loading, it notifies bsc_tables to use hasp_tables to help with the line protocol.

Details for binding bsc_tables and hasp_tables into the core image are provided in the *MAM Communications*, manual "FNP Core Images" in Section 6, and map355 and bind_fnp command descriptions in Section 7. The following is a brief recapitulation of the process.

If the source segments, bsc_tables.map355 and hasp_tables.map355 do not already have object segments:

1. Produce an object segment for each via the map355 command.

2. Extract any other needed object segments for the FNP from the object archives into the working directory. For the DN6678 series FNP with more than 64K memory, the object archives are in the >ldd>mcs>object directory. For any other FNP, the object archives are in >ldd>355>object.

3. Modify the bindfile to include the two HASP modules and use the bind_fnp command to produce the core image segment to be loaded.

4. Copy this core image to the place in the storage hierarchy described by the image pathname in the Channel Definition Table (CDT), or change the specification of the core image pathname in the CMF's FNP entry.

FNP Hardware Requirements

HASP runs on any of the FNP models, DN355, DN6632, and DN6678. The only special hardware requirement is a bisync board (not sync, as might be supposed for a synchronous protocol). The option numbers for this hardware for the DN355 and DN6632 are:

DCF6015	single channel bisync up to 9600 baud, over voice-grade lines
DCF6055	single channel broad band bisync, 9600 to 50000 baud, over conditioned data lines (usually hardwired)

For the DN6678 series FNPs:

DCF6618	dual channel bisynch up to 9600 baud, over voice-grade lines
DCF6621	dual channel broad band bisync, 9600 to 72000 baud over conditioned data lines

CONFIGURATION REQUIREMENTS

If you are using the Multics system as a host, the HASP situation is much like that when Multics communicates with any other RJE station. The HASP protocol simply allows separate process control of each workstation device by providing multiplexing and demultiplexing of the data transmission. You must define the configuration of the workstation: the number of card readers, line printers, and card punches. An operator's console must also be configured, although the central system operator must enter all commands needed to control the I/O daemons driving the devices.

The basic rules are:

- You must configure one operator's console
- You can configure a maximum of eight card readers
- You can configure a combined total of eight line printers and card punches

If you are using Multics as the workstation, the device configuration rules are the same; Multics simulates these devices. However, the simulated operator's console is used to establish the identity of the workstation with the remote host. It can subsequently be used to control the operation of the workstation, request status on jobs executing on the remote host, and examine the queues of output files waiting for transmission to Multics. The Multics workstation configuration must additionally meet any requirements of the remote host.

DEFINITION OF HASP CHANNELS

Whether Multics is acting as host or workstation, you need to define a HASP multiplexer channel in the CMF. A sample excerpt from a CMF is shown below. The comments show the correspondence between the multiplexer hierarchy defined in the CMF and the workstation configuration.

```
name: a.h014;                    /* the workstation itself */
 service: multiplexer;
 multiplexer_type: hasp;
 line_type: BSC;
 baud: 4800;
 terminal_type: HASP_WORKSTATION;

name: a.h014.opr;               /* entry for operator's console */
 service: slave;

name: a.h014.rdr1-a.h014.rdr3;   /* entry for 3 card readers */
 service: slave;

name: a.h014.prt1-a.h014.prt5;   /* entry for 5 line printers */
 service: slave;

name: a.h014.pun1;              /* entry for 1 card punch */
 service: slave;
```

The above configuration defines a remote workstation with an operator's console, three card readers, five line printers, and a single card punch.

Major Channel

In the CMF, the specified name of the major channel (e.g., a.h014) is the unique channel name of the form x.hNNN, where x is the FNP identifier, hN is the high-speed line adaptor identifier (h0, h1, or h2) and NN is the two-digit channel number identifying a subchannel of the specified HSLA.

For the major channel, the service (multiplexer), multiplexer_type (hasp), and line_type (BSC) must be specified exactly as shown. The baud statement specifies the baud rate of the channel. Baud rate must be compatible with the FNP line adapter type and can be: 2400, 4800, 7200, 9600, 19200, 40800, 50000, or 72000. The terminal_type, whose name is arbitrarily chosen, is used to distinguish between a connection to a remote workstation and one to a host and to specify options controlling the connection. These options are specified in the Terminal Type File (TTF) where the terminal type, only named here in the CMF multiplexer entry, is actually defined. Since the standard Multics TTF contains no HASP multiplexer terminal types, you must name and define your own. That is described below in "Multiplexer Terminal Types." In this manual's examples, the terminal_type names chosen indicate what Multics is "talking to," i.e., Multics is acting as host system in the example above and "talking to" a HASP workstation. This convention is followed wherever discretionary names are allowed.

Subchannels

The names of the subchannels (a.h014.*) must be created according to the following rules:

- An operator's console must be included in every multiplexer. The console is the subchannel whose final component name is "opr."

- A maximum of eight card readers are permitted. A card reader is a subchannel whose final component name is of the form "rdrN", where N is a single digit between one and eight.

- The combined number of line printers and card punches must not exceed eight. For example, a multiplexer may have eight line printers and no card punches, or three line printers and five card punches; however, a multiplexer cannot have five line printers and five card punches. A line printer is a subchannel whose final component name is of the form "prtN", where N is a single digit between one and eight. A card punch is a subchannel whose final component name is of the form "punN", where N is also a digit between one and eight.

  NOTE: Due to a restriction in the HASP communications protocol, a multiplexer cannot contain a line printer and card punch whose device addresses total nine. For example, "prt4" and "pun5" can not be configured in the same multiplexer.

  You must specify a service type of slave for every subchannel that an I/O daemon driver directly attaches.

The *MAM-Communications* manual gives a detailed description of the CMF.

Once you have made the required entries to the CMF:

1. Convert the CMF to the CDT via the cv_cmf command.

2. Install the resulting CDT via the install command.

If you edited the FNP entry's image pathname in the CMF, this new CDT allows the FNP to use the new core image the next time the FNP is loaded, either automatically at system startup or by means of the load_mpx operator command.

MULTIPLEXER TERMINAL TYPES

All terminal types, including multiplexer terminal types, are defined in the TTF. The following describes how to define the HASP multiplexer terminal type.

The Multics HASP support must distinguish between communicating with a workstation and a host. Additionally, the software must also be aware of certain characteristics of the communications channel (e.g., maximum message block length, maximum negative acknowledgments).

The configuration information and specification of options for a HASP multiplexer channel is supplied by using the value of the additional_info keyword of the multiplexer channel's terminal type. The additional_info value is formatted as a series of parameter assignments, separated by spaces or commas. Each parameter assignment has the following form:

```
<parameter_type>=<parameter_value>
```

A sample TTF entry for a multiplexer terminal type (named HASP_HOST here, indicating Multics is simulating a workstation) might appear as follows:

```
terminal_type: HASP_HOST;
 additional_info: "type=host, block_size=512, signon_mode=no";
```

The complete set of parameter types and values that can be specified for additional_info is shown below (defaults are provided for all omitted parameters).

type: type
value: workstation or host
default: workstation
meaning: specifies whether this multiplexer is connected to a remote workstation when Multics serves as host (type=workstation), or a remote host when Multics simulates a workstation (type=host).

type: block_size
value: 400 through 1017
default: 400
meaning: specifies the maximum block size, in characters, used by Multics for all messages transmitted on the multiplexer channel. When Multics simulates a workstation, you must choose a block size compatible with the host system's requirements.

type: signon_mode
value: yes or no
default: no
meaning: when connecting to a remote host, specifies that Multics must transmit a SIGNON control record to identify Multics to the remote host before any data can be transmitted; this parameter is ignored when connecting to a workstation. See "Definition of a HASP Workstation Simulator" in Section 3.

type: multileave_mode
value: yes or no
default: yes
meaning: specifies that blocks transmitted by Multics can contain records from more than one device. This parameter may need to be "no" for communications with some workstations (e.g., the workstation cannot accept a block containing records for two printers simultaneously).

type: suspend_all_mode
value: yes or no
default: no
meaning: specifies that when Multics wishes to temporarily stop input for a specific device (flow control), the system will instead request that all input be suspended. This parameter may need to be "yes" for communications with some workstations (e.g., an IBM System/370 running the HASP workstation simulation option of RSCS under VM/370).

type: rts_mode
value: yes or no
default: yes if type=host; no if type=workstation
meaning: specifies whether Multics requests permission (request to send) from the remote host or workstation before transmitting data on other than the operator's console.

type: connect_timeout
value: 1 through 60 or none
default: 30
meaning: when connecting to a remote host, specifies the period (in seconds) to attempt the initial connection sequence after the physical connection is established; when connecting to a remote workstation, specifies the period to wait for the initial connection sequence after the physical connection is established. If the string "none" is used, the multiplexer either awaits or sends the initial connection sequence indefinitely as appropriate. It is recommended that you specify "none" for a hardwired connection (no operator intervention).

type: receive_timeout
value: 1 through 60
default: 3
meaning: specifies the period (in seconds) to wait for a message to be received from the remote workstation/host before assuming that the last block transmitted by Multics was lost and should be retransmitted.

type: transmit_timeout
value: 1 through 60
default: 2
meaning: when connecting to a remote host, specifies the period (in seconds) for the FNP to wait for a block from Multics to be transmitted before automatically acknowledging the last block received from the host; this parameter is ignored when connecting to a remote workstation.

type: max_naks
value: 5 through 100
default: 10
meaning: specifies the maximum number of consecutive NAKs that can be transmitted or received by the FNP on the communications channel without an intervening block before aborting the connection. You should set this higher for systems slow to acknowledge the connection.

type: max_device_input_records
value: 3 through 30
default: 6
meaning: specifies the maximum number of records that the multiplexer holds as input for a subchannel before requesting the remote system or workstation to suspend transmission for that subchannel. For optimum operation, this parameter should be given a value that is larger than the average number of records in an input block as reported by channel_comm_meters for the multiplexer channel.

Once you have made the required entries to the TTF:

1. Convert the TTF to the Terminal Type Table (TTT) via the cv_ttf command.

2. Install the resulting TTT via the install command.

Data Translation

All data transmitted over a HASP subchannel must be translated to the character set, normally EBCDIC, of the remote host or workstation and formatted according to the rigid requirements of the HASP communications protocol.

These functions are performed by the hasp_host_ and hasp_workstation_ I/O modules (described in Section 6). A terminal type can be specified in the attach descriptions of these I/O modules to define the remote host's or workstation's character set. If one is specified, it must be a terminal type defined in the TTT with both an input and output translation table for data conversion to/from ASCII and the remote host system's or workstation's character set. You can use an already-defined terminal type or define one yourself that meets this requirement and install it as described above. If translating to/from EBCDIC, note that EBCDIC has no equivalents for "[" and "]". Different manufacturers use different characters to translate to and from these characters. Therefore, take care with RJE and card input that uses active functions. The *Multics Programmer's Reference* manual provides additional information for defining terminal types.

If a terminal type is not to be specified, the remote host system's or workstation's character set is assumed to be EBCDIC; standard Multics subroutines are used for data conversion. Again, these may not handle "[" and "]" in a manner acceptable to the remote host or workstation.

FNP SPACE REQUIREMENTS

For an FNP configured with 32K of memory, the bsc_tables and hasp_tables modules require 2358 and 980 words, respectively. You should step through the calculations given in "Multics Communications System Memory Configuration" of *MAM-Communications* if necessary to determine whether you have sufficient space for supporting HASP.

The formula for determining the memory required to support the HASP protocol is number of echoplexed HASP channels multiplied by memory per channel. HASP channels require 54 words in the software communications region, 42 words for the terminal information block (TIB), 2 words for the TIB table entry, 32 words for the TIB extension buffer, and 2 + block_size/2 for each HASP I/O buffer (where block_size is specified by the TTF entry of the multiplexer and there are 2 words of buffer overhead).

For a DN6670 configured with at least 64K of memory, the bsc_tables and hasp_tables modules require 2440 and 986 words, respectively.

The same formula for determining memory to support the HASP protocol is used. Memory needed per HASP channel is 34 words; you should multiply by the number of echoplexed HASP channels. The size in words of one HASP I/O buffer is 2 + block_size/2 (where block_size is specified by the TTF entry of the multiplexer and there are 2 words of buffer overhead). Again, see *MAM-Communications* if detailed calculations and memory figure breakdown are required.

SPACE REQUIREMENTS IN TTY_BUF

The space used in tty_buf is occupied by various data bases that are used to maintain proper control of the Multics communication management system.

Static space is allocated in tty_buf for output DCW lists for each FNP. This occupies 128 words for each loaded FNP (8 DCW lists * 16 words per DCW list).

The logical channel table (LCT) is also maintained in tty_buf. Its size is determined by the number of channels configured in the CMF. It is defined by the structures in the lct.incl.pl1 include file. There is a 16-word header at the beginning of the LCT. Each channel (multiplexer or subchannel) requires a 32-word entry (LCTE).

A physical channel block (PCB) is configured for each channel of each FNP (names that match *.*, including multiplexers, but not subchannels. of multiplexers), eight words per channel.

A wired terminal control block (WTCB) of 20 words is allocated in tty_buf for each subchannel.

I/O buffers are dynamically created and deleted as needed. Each communications protocol makes different demands on tty_buf space. (See "Dynamic Storage in tty_buf" below.)

Subchannel and Multiplexer Channel Static Data Requirements

The HASP multiplexer type requires a data base defined by the "hmd" structure in the hasp_mpx_data.incl.pl1 include file.

```
PCB       8
LCTE      32
HMD       56
Total     96 words
```

Each HASP subchannel of a HASP multiplexer requires a data base defined by the "hste" structure in the hasp_mpx_data.incl.pl1 include file.

```
LCTE      32
HSTE      30
WTCB      20
Total     82 words
```

CALCULATION OF STATIC STORAGE IN TTY_BUF

The following chart outlines the required data needed to find out how much static storage is required in tty_buf. The data is obtained from the CDT.

type of channel	number	X	memory/ channel	=	memory required
tty_buf header			-		72
DCW lists			128		
LCT header			-		16
spare channel count in CDT			32		
FNP subchannels			60		
HASP multiplexers			96		
HASP subchannels			82		
			Total		

To get the total figure, other channels types must be taken into account. See *MAM-Communications* for the details on them.

Dynamic Storage in tty_buf

It is difficult to predict what demands the users of a system will place on the available I/O buffer space in tty_buf. This has to be measured with system_comm_meters. It has been found that tty_buf should not run more than 80% full to handle peak loads best.

Delay queues are also dynamically allocated in tty_buf and can crash the system if there is no room for them. These are transactions that the host is requesting the FNP to perform. If there is no mailbox to put the transaction into, it is entered into the delay queue for later processing.

HASP I/O BUFFER SPACE

Some comments can be made about the HASP protocol's handling of its I/O buffer space in tty_buf. Only an actively operating line will require I/O buffer space in tty_buf. The following calculations should only be performed on the number of channels expected to be active at any one time.

The size of HASP buffers in tty_buf is controlled by the block_size parameter in the TTF entry of the multiplexer. An active HASP multiplexer will most likely have three blocks in tty_buf: two for output and one for input.

```
HASP-10 = 3 * ttf_block_size
```

ACS SEGMENTS

You must create an ACS segment for each subchannel of the HASP multiplexer channel. Once you have done so:

1. Give the daemon process that will attach that subchannel rw access to the ACS.

2. Give the attaching process the dialok attribute in the Project Master File (PMF). It is recommended that the process that attaches the simulated operator's console not be registered on the SysDaemon project, since that may confer undesirable access to too much of the system.

SECTION 3

ADMINISTRATIVE SETUP - I/O DAEMON

I/O DAEMON TABLE

A HASP workstation is composed of card readers, card punches, line printers, and an operator's console. Each device is configured as a separate subchannel of the physical communications channel defined in the CMF as the HASP multiplexer channel. Each device except the operator's console is, in addition, configured as an I/O daemon in the I/O daemon table. The I/O daemon table defines the devices and Request_types to be used with the I/O daemons. The table is a source file that consists of a sequence of statements and substatements defining and describing each device and Request_type. The *Bulk I/O* manual provides a full description of the I/O daemon table, the iod_table_compiler and the create_daemon_queues commands; this section presents only the definitions required for HASP operation.

The pathname of the source of the I/O daemon table is usually:

>daemon_dir_dir>io_daemon_dir>iod_tables.iodt

WORKSTATION STRUCTURE - MULTICS AS HOST

When Multics is the host system, the remote_driver_ I/O daemon driver module controls the remote workstation devices. You must:

- Define a major device with one minor device for each workstation device in the system iod_tables.

Configure each device as a separate Type II I/O daemon (since the workstation has no input device even though an operator's console must be specified in the CMF). A Type II workstation is initialized on a dedicated communications line as a predefined station. The workstation can be identified only by the line it dials into; no Line statements are associated with Type II stations. Thus, the line substatement must specify exactly the appropriate subchannel of the HASP multiplexed channel on which the remote station will be connected. An args substatement must specify use of the hasp_workstation_ terminal I/O module. In each driver's args substatement, the slave parameter must be:

slave=no

Sample iod_tables Definition

The iod_tables entries for a remote workstation (Multics as host) with a card reader, two line printers, and a card punch, follows.

```
Device:                 vax_rdr1;
  line:                 a.h014.rdr1;
  driver_module:        remote_driver_;
  args:                 "station= vax, slave= no,
                        desc= -terminal hasp_workstation_ -comm hasp";
  minor_device:         rdr1;
    default_type:       vax_dummy:
    minor_args:         "dev=reader";

Device:                 vax_ptr1;
  line:                 a.h014.prt1;
  driver_module:        remote_driver_;
  args:                 "station= vax, slave= no,
                        desc= -terminal hasp_workstation_ -comm hasp";
  minor_device:         prt1;
    default_type:       vax_prt
    minor_args:         "dev= printer";

Device:                 vax_prt2
  line:                 a.h014.prt2
  driver_module:        remote_driver_
  args:                 "station= vax, slave= no,
                        desc= -terminal hasp_workstation_ -comm hasp";
  minor_device:         prt2;
    default_type:       manuals;
    minor_args:         "dev= printer";
```

```
Device:                 vax_pun1;
  line:                 a.h014.pun1
  driver_module:        remote_driver_
  args:                 "station= vax, slave= no,
                        desc= -terminal hasp_workstation_ -comm hasp";
  minor_device:         pun1;
    default_type:       vax_pun
    minor_args:         "dev= punch;

Request_type:           vax_dummy;
  driver_userid:        HASP.SysDaemon;
  generic_type:         none;
  max_queues:           1;
  device:               vax_rdr1.rdr1

Request_type:           vax_prt
  driver_userid:        HASP.SysDaemon;
  generic_type:         printer
  device:               vax_prt1.prt1

Request_type:           manuals;
  driver_userid:        HASP.SysDaemon;
  generic_type:         printer
  max_queues:           2;
  device:               vax_prt2.prt2
  rqti_seg:             manuals_info;

Request_type:           vax_pun;
  driver_userid:        HASP.SysDaemon;
  generic_type:         punch;
  device:               vax_pun1.pun1;
```

DEVICE STATEMENT AND SUBSTATEMENTS

Device: <name>;
defines the name of a major device and denotes the beginning of a device description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces. (Our example names indicate host system and type of device.)

line: <name>;
<name> must be the subchannel name of the HASP multiplexed channel that corresponds to this device.

driver_module: <name>;
for HASP with Multics as host system, <name> must be remote_driver_.

ARGS SUBSTATEMENT KEYWORDS

The following keywords are those most likely to be needed in defining HASP iod_tables entries:

station= <station_id>
: allows the driver to accept a a non-blank station_id. The driver accepts any device dialing in on this channel as this station_id without authentication controls. All specified minor devices and default request types are used. Normally, the value is the name of the Device (i.e., the station name). This is used for a station without an input device or with a dedicated communications line.

slave= <no>
: This key is optional, since the default is "no." Only the Multics central system operator can issue commands.

desc= <attach_description>
: specifies the attach description used to attach the terminal/device I/O module. This keyword is required. The attach description must include the "-terminal hasp_workstation_" and "-comm hasp" options; the "-tty" option is provided automatically by the driver process.

MINOR DEVICE SUBSTATEMENTS

minor_device: <name>;
: defines the name of a minor device and denotes the beginning of its description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces.

default_type: <name>;
: defines the default request type for the associated minor device. The <name> must be the same as that of a Request_type statement.

MINOR_ARGS SUBSTATEMENT KEYWORDS

dev = <minor_device_type>
: specifies the device type of the minor device. This keyword is required and must be reader, printer, or punch.

REQUEST_TYPE STATEMENT AND SUBSTATEMENTS

Request_type: <name>;
defines the name of a request type and denotes the beginning of its description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces.

driver_userid: <access_name>;
defines the required person and project names for a driver of the associated request type. If omitted, defaults to IO.SysDaemon.

generic_type: <name>;
defines the generic type of this request type. For HASP, <name> should be printer, punch, or none (for the dummy request associated with a card reader). If the generic type and request type names are the same, the request type is the default for the generic type. This statement is required for a Request_type statement.

max_queues: <N>;
defines the maximum number of queues, where N can be 1, 2, 3, or 4. This substatement is optional.

device: <name>;
specifies a device that can be used to process requests of this request type. The <name> must be of the form device_name.minor_device_name, where the names are the same as that of a Device statement and its minor_device substatement, respectively. Multiple device substatements can be specified for a request type.

rqti_seg: <name>;
defines the name of the rqti segment to be used. This substatement is optional; if omitted, the driver does not look in the rqti segment directory for an rqti segment. For automatic printing without operator intervention, set the auto_go parameter for the driver in the rqti segment. See "Request Type Info Segments" below for more information.

WORKSTATION STRUCTURE - MULTICS SIMULATING WORKSTATION

When Multics is the workstation, the hasp_ws_sim_driver_ I/O daemon driver module simulates the operation of a workstation's card readers, line printers, and card punches; the hasp_host_operators_console command simulates the console. A separate process is used to simulate each device to permit all devices to operate asynchronously, thus achieving maximum throughput over the communications line.

Card decks are transmitted from Multics through the simulated card readers to the remote host system. These decks are normally jobs to be executed by the remote system. On Multics, each card deck must be contained in a segment. A Multics user requests that a deck be transmitted by issuing the enter_output_request (eor) command; a separate request type is used for each remote system.

The remote system transmits output files to Multics through the simulated line printers and card punches. By default, the simulator automatically issues printer or punch requests for these files as appropriate. However, a site may choose to have these output files placed into the system pool storage for subsequent retrieval by Multics users via the copy_cards command. To use this option, the driver process must be instructed to expect control records in each output file and the remote system must include these Multics control records to indicate which Multics user owns the file. Adding control records to an output file may involve modifications to the remote computer's operating system, the JCL of each job submitted for remote execution, the programs executed by each job, or a combination of the above. User-supplied control records required are ++IDENT and ++INPUT. (See the *Multics Programmer's Reference Manual* for a description of the format of these control records. Specific information is given under "Input and Output Facilities" and "Punched-Card Input Output and Returned Output Control Records.")

Definition of a HASP Workstation Simulator

To define a workstation simulator, you must:

- Define a major device, with exactly one minor device, for each simulated device except the operator's console in the system iod_tables. Define a request type for the submission of card decks in the system iod_tables.

- Determine if the remote system requires that a SIGNON control record be transmitted to establish the identity of the workstation. The SIGNON record is a special record defined by the HASP protocol to enable the host system to establish the identity of the workstation. Many operating systems do not require this control record, but validate the workstation in other ways. If a SIGNON record is required, its exact content must be determined for use in the hasp_host_ I/O module attach description. When the hasp_host_operators_console command is issued, also use the -signon control argument, which specifies the text of the SIGNON control record. (When Multics is the host system, it ignores any SIGNON record sent it by a workstation.)

- Determine the printer channel stops used in output files returned from the remote system and insure that the Multics request type(s) used to print those files include the appropriate logical channel stops in their RQTI segments. (See "Request Type Info Segments" below.) For example, many systems use channel stop #1 to represent the top of a page; the RQTI segments should specify "Line (1): 1;" to insure correctly formatted output.

The major device definition must include a line substatement specifying the subchannel of the simulated device; the "line: variable;" construct is not allowed. Additionally, an args substatement must be included specifying a station ID and use of the hasp_host_ terminal I/O module.

The minor device specification must include a minor_args substatement that specifies the type of device being simulated. Additional keywords can be used in this statement as described below.

Sample iod_tables Definition

The iod_tables entries to simulate a HASP workstation with a card reader, two line printers, and a card punch follows:

```
Device:                 cdc_rdr1;           /* Card reader */
  line:                 a.h014.rdr1;
  driver_module:        hasp_ws_sim_driver_;
  args:                 "station= CDC, desc= -terminal hasp_host_
                        -comm hasp";
minor_device:           rdr1;
  default_type:         cdc_jobs;
  minor_args:           "dev= reader_out";

Device:                 cdc_prt1;           /* Line printer #1 */
  line:                 a.h014.prt1;
  driver_module:        hasp_ws_sim_driver_;
  args:                 "station= CDC, desc= -terminal hasp_host_
                        -comm hasp";
minor_device:           prt1;
  default_type:         dummy;
  minor_args:           "dev= printer_in, request_type= cdc_output";
```

```
Device:                 cdc_prt2;       /* Line printer #2 */
  line:                 a.h014.prt2;
  driver_module:        hasp_ws_sim_driver_;
  args:                 "station= CDC, desc= -terminal hasp_host_
                        -comm hasp";
minor_device:           prt2;
  default_type:         dummy;
  minor_args:           "dev= printer_in, auto_queue= no";

Device:                 cdc_pun1;       /* Card punch */
  line:                 a.h014.pun1;
  driver_module:        hasp_ws_sim_driver_;
  args:                 "station= CDC, desc= -terminal hasp_host_
                        -comm hasp";
minor_device:           pun1;
  default_type:         dummy;
  minor_args:           "dev= punch_in";

Request_type:           cdc_jobs; /* Request type for submitting */
  generic_type          punch;  /* card decks to remote CDC system */
  max_queues:           1;
  device:               cdc_rdr1.rdr1;

Request_type:           dummy;      /* Required by printers and */
  generic_type:         none;       /* punches to avoid errors from */
  max_queues:           1;          /* iod_tables_compiler */
  device:               cdc_prt1.prt1;
  device:               cdc_prt2.prt2;
  device:               cdc_pun1.pun1;
```

DEVICE STATEMENT AND SUBSTATEMENTS

Device: <name>;
defines the name of a major device and denotes the beginning of a device description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces. (Our example names indicate host system and type of device.)

line: <name>;
<name> must be the subchannel name of the HASP multiplexed channel that corresponds to this simulated device.

driver_module: <name>;
For HASP with Multics simulating a workstation, <name> must be hasp_ws_sim_driver_.

ARGS SUBSTATEMENT KEYWORDS

These define the characteristics of this device. The following are required for HASP applications.

station= <station_id>

identifier placed on the head sheet or header cards of returned output files when said files are printed/punched automatically. This keyword is required; the same value should be used for all devices of a workstation simulator.

desc= <attach_description>

specifies the attach description used to attach the terminal/device I/O module. This keyword is required. The attach description must include the "-terminal hasp_host_" and "-comm hasp" options; the "-tty" option is provided automatically by the driver process. If the remote system requires a SIGNON record, the "-signon" option must be included for all devices of the workstation.

MINOR DEVICE SUBSTATEMENTS

minor_device: <name>;

defines the name of a minor device and denotes the beginning of its description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces.

default_type: <name>;

defines the default request type for the associated minor device. The <name> must be the same as that of a Request_type statement.

MINOR_ARGS SUBSTATEMENT KEYWORDS

dev= <device_type>

specifies the type of device being simulated by this driver process. This keyword is required. The acceptable values for device_type are:

reader_out

simulates a card reader for sending card decks to the remote system.

printer_in

simulates a line printer for receiving output files from the remote system.

punch_in
simulates a card punch for receiving card decks from the remote system.

auto_receive= <yes/no>
specifies the mode of operation of this driver whenever communication is established with the remote system. The possible choices are "yes" to wait for output files from the remote system (especially useful with hardwired connections), or "no" to listen for I/O daemon commands from the operator.

This keyword cannot be given if "dev= reader_out" is specified. This keyword is optional; the default value is "no" (listen for I/O daemon commands).

auto_queue= <yes/no>
specifies how output files received by this driver are handled. The possible choices are "yes" to queue the files for local printing/punching, or "no" to scan them for Multics control records and store them in system pool storage for online perusal.

This keyword cannot be given if "dev= reader_out" is specified. This keyword is optional; the default value is "yes" (automatically queue output files).

request_type= <rqt_name>
rqt= <rqt_name>
specifies the Multics request type to be used for automatically printing or punching output files. The request type specified must be of generic type "printer" if "dev= printer_in" is specified or generic type "punch" if "dev= punch_in" is specified; this keyword cannot be given if "dev= reader_out" is specified. This keyword is optional; the default request type used is the default specified for the appropriate generic type.

REQUEST_TYPE STATEMENT AND SUBSTATEMENTS

Request_type: <name>;
defines the name of a request type and denotes the beginning of its description. Any <name> can be chosen; it can be a maximum of 24 characters and cannot contain periods or spaces.

driver_userid: <access_name>;
defines the required person and project names for a driver of the associated request type. If omitted, defaults to IO.SysDaemon.

generic_type: <name>;
defines the generic type of this request type. If the generic type and request type names are the same, the request type is the default for the generic type. This statement is required for a Request_type statement.

max_queues: <N>;
defines the maximum number of queues, where N can be 1, 2, 3, or 4. This substatement is optional.

device: <name>;
specifies a device that can be used to process requests of this request type. The <name> must be the same as that of a Device Statement. Multiple device substatements can be specified for a request type.

rqti_seg: <name>;
defines the name of the rqti segment to be used. This substatement is optional; if omitted, no driver looks in the rqti segment directory for an rqti segment. For automatic printing without operator intervention, set the auto_go parameter for the driver in the rqti segment. See below for more information.

REQUEST TYPE INFO SEGMENTS

Each printer request type can have an optional request type info segment (rqti segment) associated with it that defines the physical paper characteristics, the logical VFU channel stops, and some additional driver control data. This is true whether Multics is acting as host or simulating a workstation. Special forms should have a specific request type and thus a separate set of channel stops. The channel stops are set only during driver initialization and remain constant for all requests done by the driver.

In addition, a site may wish to use the request type feature to group requests that use the same VFU tape, regardless of what preprinted form stock is needed for the request. By using the "^auto_print" driver mode, the operator can run requests associated with a given VFU tape (request type) in sequence and change the form stock on the printer to meet the needs of each request.

Printers that have firmware loadable VFC images are loaded by the driver during driver initialization (the paper may have to be realigned by the operator). For printers that use punched paper VFU tapes, the physical VFU tape for the request type must be mounted on the printer at the time the driver is initialized. The driver indicates the number of lines-per-page and the lines-per-inch switch setting that the operator should use.

The size of the head and tail sheets is set automatically to the physical dimensions of the paper as defined in the request type info segment.

The directory named >ddd>iod>rqt_info_segs must give sma access to the administrator and s to all other users. The initial ACL for segments must be set to rw for the administrator and r to all other users. AIM access, for those sites using the access isolation mechanism, should be system_low (the default). This directory contains all request type info segments. If a single segment describes the paper characteristics for more than one request type, added names can be used in place of separate identical segments.

Info segments are only required for printer request types that have the rqti_seg substatement in the iod_tables. When no rqti segment is used, the defaults described for the cv_prt_rqti command are used (see "Syntax for the Request Type Info Source Segment" below). Create printer rqti segments via the cv_prt_rqti table conversion command. A sample source file is shown in "Example of a Request Type Info Segment" below. The contents of an rqti segment can be printed by the display_prt_rqti command. This command formats its output so that when directed to a file, the file can be used as input to the cv_prt_rqti command.

Syntax For The Request Type Info Source Segment

The request type info source segment contains keywords that define certain values put into the request type info segment. The general syntax is of the form:

keyword: <value>;

where the keyword defines a parameter to be set, and the <value> defines what the value of the parameter is.

The keywords and the values acceptable to the cv_prt_rqti command are listed below. Those of special interest for HASP workstations are described in detail. More information on the others is available in the *Bulk I/O* manual.

driver_attributes: [^] auto_go {, [^] meter} >;
establishes some operating parameters for the driver. There are two values defined: auto_go and meter. Each value can be preceded by the character "^" to negate the parameter. The driver_attributes keyword is optional (the default is ^auto_go,^meter).

The auto_go value is used to make the central site or remote printer driver request service from the coordinator immediately after initialization without asking for a go command. For printers on Multics HASP workstations, the auto_go value is particularly useful as a means of starting or resuming the processing of print requests without operator intervention.

The meter value is used to tell the driver to maintain internal metering data about its operation. (Note: metering is done according to the driver module design and not all driver modules implement metering.)

driver_wait_time: <number>;
value is a decimal number between 30 and 300; default is 30 seconds.

prt_control: [^] value{, [^] value...};
value is force_ctl_char; default is ^force_ctl_char. A Multics workstation cannot use force_ctl_char mode; it is honored only by the remote_driver_ driver module.

message: <"string">;
message character string must not be longer than 256 characters.

paper_length: <number>;
Value is a decimal number between 10 and 127; default is 66 lines.

paper_width: <number>;
Value is a decimal number between 1 and 136; default is 136 character positions.

lines_per_inch: <6/8>;
The default is 6 lines.

line(<line_no>): <ch_1,ch_2,ch_3,...,ch_n>;

the line keyword is optional. There can be one line keyword for each line from 1 to the paper_length. The line keyword specifies which logical VFU channels are defined to stop at <line_no>. There can be 1 to 16 channel stops for any given line, each ch_i is a number between 1 and 16.

For example:

```
line(20): 1,5,11;
```

specifies that a slew to channels 1, 5, or 11 causes the printer to stop at the beginning of line 20.

NOTE: Line 1 is always defined as the form feed position. Typically the operator positions line 1 at the fourth printable line on a page.

end;

this keyword is required. The end keyword has no value. It specifies the end of the request type info source segment.

Example of a Request Type Info Source Segment

```
/* SAMPLE SOURCE FILE FOR A PRINTER REQUEST TYPE INFO SEGMENT  */
/* Source file:    invoices.rqti                                */
/* Data segment:   invoices                                     */

/* The first two keywords apply to the header data only.  */

driver_attributes:  ^auto_go;  /* the default */
driver_wait_time:   30;   /* number of seconds driver will   */
                          /* wait before asking coord again  */


prt_control:        force_ctl_char;

/* Message to the operator during driver initialization      */

message:
"For the invoices, use VFU tape number 12.
The form stock is in storage bins 22, 23, and 24.";
```

```
/* Physical Paper Info */

/* The form stock is only 80 print positions wide and
   72 lines per page at 8 lines per inch                    */

paper_width:        80;      /* default is 136 */
paper_length:       72;      /* default is 66 */
lines_per_inch:     8;       /* default is 6 */


/* Channel Stops */

/* The logical channel stops are defined as follows:        */

line(1):            1;   /* channel 1 is top of form */
line(3):            4;   /* chan 4 is the address line */
line(12):           7;   /* chan 7 is the first entry line */
line(60):           7;   /* and is also the bottom line */

end;
```

STEPS AFTER EDITING I/O DAEMON TABLE

Once you have edited the I/O daemon table source:

1. Compile it via the iod_tables_compiler command. For convenience, store the pre-compiled version in the same directory as the compiled version with the name iod_tables.iodt.

2. Create the I/O daemon queues via the create_daemon_queues command.

3. Register the I/O daemons in the PMF, if necessary; the initproc entry's pathname should be iod_overseer_ for each daemon, and the ^vinitproc and dialok attributes should be assigned for each. Convert the PMF to a Project Definition Table (PDT) via the cv_pmf command and install the new PDT via the install command.

4. Set up message coordinator routings to handle the HASP I/O daemon. In order for the daemon to issue messages and receive commands, you must create the input and output message segments, and establish the message coordinator sources and virtual consoles. To create the input and output message segments, use:

   ```
   create >sc1>(input_SOURCE_name output_SOURCE_name).message
   ```

   Establish the message coordinator sources and virtual consoles via the define and route commands. Usually the two commands are entered in the system_start_up.ec file. HASP messages can be sent to an actual terminal, a log file, or to both. To send messages to a terminal, you must register a message coordinator channel for the terminal in the CMF (see *MAM-Communications*) and issue an accept command in addition to the define and route commands.

The define command defines the virtual console that will receive messages from your HASP daemon. Its format is:

```
define VCONS TYPE DEST
```

where VCONS is the virtual console's name, TYPE is tty, log, or sink; and DEST is the destination. For example:

```
sc_command define iod tty b.h200
```

defines a virtual console named iod that forwards all output sent to it to the terminal whose channel is b.h200. The sc_command is used to add operator commands to an ec file.

The command line:

```
sc_command define iolog log iolog
```

defines a virtual console named iolog that forwards all output sent to it to the log file named >sc1>iolog.

The route command sends output from the HASP daemon to the designated virtual console. You must issue a route command for the user_i/o, error_i/o, and log_i/o streams of the drivers. The format is:

```
route SOURCE STREAM VCONS
```

where SOURCE is the name of the output source; STREAM is the name of the stream carrying the output; VCONS is the name of the virtual console to which output is to be routed. For example:

```
sc_command route cdc_prt user_i/o iod
sc_command route cdc_prt error_i/o *iod
sc_command route cdc_prt log_i/o iod
sc_command route cdc_prt user_i/o iolog
sc_command route cdc_prt error_i/o *iolog
sc_command route cdc_prt log_i/o iolog
```

In the above, output from the HASP daemon using the cdc_prt source is routed to the two virtual consoles, iod (a terminal) and iolog (a log file) defined in the previous examples. The asterisk before the virtual console name for the error_i/o stream causes the terminal to issue an audible alarm whenever error messages are issued.

SECTION 4

OPERATING A HASP WORKSTATION

WORKSTATION INITIALIZATION

The following describes the operation of a remote HASP workstation when Multics is acting as the host system. To start a remote HASP workstation connected to a Multics host:

- If necessary, issue the initializer "load_mpx" command described in the MOH to cause the HASP multiplexer channel to wait for a connection.

- Login the I/O coordinator process (if it's not done automatically at system startup) usually from the initializer terminal via the:

  login IO SysDaemon <source_id>

  command line. (See the *Bulk I/O* manual's Section 3 for details.) Also, specify the authorization, if necessary, when AIM is in use.

- Complete the physical connection to the remote workstation for dial-up connections.

- Once the IO.SysDaemon is logged in, the system prints "Enter command: coordinator or driver." Issue the coordinator command (by typing cordinator or the shortname coord). When the initialization is finished ("I/O coordinator initialized" message appears), the coordinator is at command level and ready to accept drivers.

- Login each of the driver processes for the workstation devices in the same way you logged in the coordinator. The driver login identifier is usually IO.SysDaemon. When the system prints "Enter command: coordinator or driver" issue the driver command (by typing driver). The driver prints its version number and then prints:

  Enter command or device/request_type:

  Type the name of a device or a command:

  <device_name> {<request_type>}
  runs the given device

  print_devices
  prints device names and request types

  logout
  logs out the driver

  After initialization is complete, the driver prints:

  <device> driver ready at <time>
  Enter command:

- Enter any commands required to ready all the devices of the workstation.

- Enter the go command to begin processing requests.

SIMULATOR INITIALIZATION

The following describes the operation of a HASP workstation when Multics is simulating a workstation communicating with a remote host system. To start a HASP workstation simulator:

- If necessary, issue the initializer "load_mpx" command described in the MOH to cause the HASP multiplexer channel to wait for a connection.

- Login the process that is to run the simulated operator's console of the workstation, and issue the hasp_host_operators_console (hhoc) command (described in Section 5) to wait for the connection to be completed. If the remote system requires a SIGNON record as part of the connection procedure, include the "-signon" option on the hhoc command line.

- Complete the physical connection to the remote system for dial-up connections.

- When the process running the operator's console prints the message "Input:" indicating that the physical connection is established, perform any logon sequence required to identify the workstation to the remote system. The exact sequence used, if any, should be determined from the remote system's administrative staff.

- Login each of the driver processes for the other simulated devices. The sequence used to login a driver process is the same as that described above in "Workstation Initialization".

- On the terminal of the process running the operator's console, issue any commands to the remote system required to ready all the devices of the workstation.

- For each driver process running a simulated card reader, issue commands:

  ```
  ready
  go
  ```

  These commands will start the transmission of card decks to the remote system.

- Issue the receive command for each driver process running a simulated line printer or card punch. This command causes these drivers to wait for output files to be sent by the remote system. As each output file is received, it is processed according to the specifications given in the minor_args substatement of the driver.

SECTION 5

COMMANDS FOR DEVICE/DRIVERS

This section describes the I/O daemon commands that can be used by the driver processes. They are entered at the operator's console; the hasp_host_operators_console command allows simulation of the operators console for a Multics workstation. When Multics serves as the workstation, the printer and punch simulators have only a subset of these commands available. This subset is:

auto_queue
clean_pool
hasp_host_operators_console
help
hold
inactive_limit
logout
new_device
receive
reinit
release
request_type
start
status
x

When Multics serves as host, you do not use the auto_queue, hasp_host_operators_console, receive, or request_type commands.

Name: auto_queue

The auto_queue command controls whether output files received by this driver are automatically printed or punched locally, or scanned for Multics control records and placed in system pool storage for online perusal.

Usage

auto_queue <switch_value>

where:

switch_value
must be chosen from:

yes
automatically queue the files for printing/punching; do not scan for control records.

no
scan the output files for Multics control records and store them in system pool storage for online perusal; do not automatically queue files for printing/punching.

Name: auto_start_delay

The auto_start_delay command displays or sets the length of time the driver waits to issue the start command automatically after receiving a quit signal. An automatic start is cancelled if command input is received.

Usage

auto_start_delay {N}

where N is the desired delay time in seconds. N must be at least 30 seconds. The default delay time is 60 seconds. When no argument is given, the current delay time is displayed.

Name: banner_bars

The banner_bars device-specific driver command is used by printer drivers to establish how the separator bars at the bottom of the head sheet are to be printed. Printers that can overstrike should use "double" (this is the default). Other printers should use single.

Usage

banner_bars {minor_device} {arg}

where minor_device is a minor device name (as shown by the status command) and is required if there is more than one printer minor device; arg can be one of the following separator types:

double
: overstrikes each separator line.

single
: single strikes each separator line.

none
: suppresses separator lines.

-print
: if arg is not given, or if a single arg "-print", is given, the current value is printed.

Name: banner_type

The banner_type device-specific driver command is used by printer drivers to change the information printed on the front and back of each copy of a request.

Usage

banner_type {minor_device} {key}

where minor_device is a minor device name (as shown by the status command) and is required if there is more than one printer minor device; key must be one of the following:

standard
: prints the normal head and tail sheets.

none
: prints nothing except the separator bars, if required (according to the banner_bars command).

brief
: prints a short version of the head and tail sheets.

-print
: if arg is not given, or if a single arg "-print", is given, the current value is printed.

Name: cancel

The cancel command terminates the request that the driver is currently processing. The request is not placed in the coordinator's saved list and thus cannot be restarted later. This command is only valid after a quit signal, or at request command level.

After completing the command, the driver looks for another request to process. (In step mode, it returns to command level.)

Usage

cancel

Name: clean_pool

The clean_pool device-specific driver command applies to drivers that can read user card decks. It allows the operator to delete all segments in the system card pool that have been there more than a specified number of days. This command is not available for slave terminals.

Usage

clean_pool N

where N is the maximum length of time in days for segments to be retained in the system card pool. All segments that have been in the card pool more than that number of days are deleted. N must be a decimal number greater than zero.

Name: copy

The copy device-specific driver command sets the copy number of the next copy of the current request to the value specified. This command is used only at request command level.

Usage

copy N

where N is a decimal integer between 1 and the number of copies requested by the user.

Name: ctl_term

The ctl_term command applies only to a control terminal (if attached). It allows the operator to specify the format of printed output.

One of the primary functions of the control terminal is to print information about each request processed, to aid in separating the output, and to ensure proper accountability of output generated by the driver. The site can use preprinted forms for this purpose (see "Using Preprinted Accountability Forms on the Control Terminal" in the *Bulk I/O* manual. In this case, alignment of the data on the form is very important. Generally a terminal that supports vertical tab and formfeed control characters is used to ensure alignment. However, this command allows the operator to request that the software simulate the action of formfeed control characters if the terminal does not provide this hardware support.

Usage

ctl_term arg

where arg falls into one of two classes: general control or simulation control (some arguments require an additional value to define the requested action):

general control

form_type STR
: specifies the format program to be used to format the data printed on the control terminal. If STR is "default", the form_type is set to the default format.

detach
: discontinues the use of the control terminal. This argument is restricted to the master terminal and is not reversible unless the reinit command is given.

simulation control

simulate
: sets the driver to simulate formfeeds by software. (This argument is not reversible even by the reinit command.)

page_length N
: sets the number of lines per logical page to N. This controls the forward spacing needed to go to the top of the form.

aligned
: indicates that the forms are aligned for the purpose of form feed control. (A sample form can be printed by the sample_form command.)

Name: defer

The defer command sends the current request back to its queue marked as deferred. It is only issued from quit command level or request command level.

Usage

defer

Notes

Requests are automatically deferred when the requested line length of the device exceeds the physical line length, or when the estimated processing time of a request exceeds the operator-defined limit (see the defer_time command below).

A deferred request is reprocessed when the driver is given the restart_q command or when the coordinator is next initialized.

Name: defer_time

The defer_time command sets or displays the current time limit for automatically deferring requests.

Usage

defer_time {minor_device} {N}

where:

1. minor_device
   is the name of the minor device for which the time should be set or displayed. It is optional for drivers that have only one minor device (e.g., the central site printer). If specified, this argument must be the first argument.

2. N
   sets a new defer time in minutes, with a precision of tenths (e.g., 1.5 is one minute, 30 seconds). A time of zero indicates that infinite time is allowed. If N is not given, the current defer time and driver output rate are displayed.

Name: go

The go command makes the driver look for requests to process. If no requests are currently available, the driver asks the coordinator for a request for each "ready" device. These requests are processed as soon as they are provided by the coordinator. (This command can not be used at request command level or immediately following a quit signal.)

Usage

go {N}

where N is the number of requests processed before the driver returns to command level. If N is not specified, the driver continues to process requests and does not return to command level until requested by the operator.

Name: halt

The halt command provides the reverse function of the ready command. It places the device or each of the specified minor devices in the inactive state. The driver does not ask the coordinator for any further requests for a halted device. However, the coordinator may have already supplied a "pending request" for the halted device. In this case, any pending request is processed immediately after the device has been halted (except when the command has been issued following a quit signal).

Usage

halt dev1 ... dev_ {-control_arg}

where:

1. dev_
   is the name of a device, or minor device in the case of a multifunction device, that is to be placed in the inactive state. The device names that can be used are those printed out by the status command.

2. control_arg
   can be -all or -a to halt all devices. If the -all control argument is used, no device names need be given. No control argument is required if there is only one device for the driver. If there are multiple minor devices, the operator must specify the ones to be made inactive or else must specify -all to halt all minor devices.

hasp_host_operators_console

Name: hasp_host_operators_console, hhoc

The hasp_host_operators_console command simulates the operation of the operator's console of a HASP workstation. The operator's console is used to identify a Multics workstation to a remote system, to issue commands governing the operation of the workstation, and to receive status information from the remote system.

Usage

hhoc tty_channel {control_args} {attach_arguments}

where:

1. tty_channel
   is the name of the terminal channel to be attached as the operator's console. This channel must be configured as the console subchannel of a HASP multiplexer channel (e.g., a.h014.opr).

2. control_args
   can be chosen from the following:

   -signon STR
   specifies that the remote host requires a SIGNON record to be transmitted before data transmission can occur. STR is the text of the control record; it may be up to 80 characters in length. Before transmission it is translated to uppercase and the remote system's character set.

   -no_signon
   specifies that the remote host does not require a SIGNON record. (Default)

3. attach_arguments
   are options acceptable to the hasp_host_ I/O module. This command supplies the -comm, -tty, and -device options automatically; these options need not be given on the command line. (See Section 6 for a description of the hasp_host_ I/O module.)

Notes

If the remote system requires a SIGNON, the -signon option should be supplied on the command line specifying the exact SIGNON record to be transmitted. For example, the command line:

hhoc a.h014.opr -signon "/*SIGNON REMOTE7"

attaches the a.h014.opr channel as the operator's console of a remote IBM system expecting a connection from the workstation named REMOTE7.

After attaching the channel specified on the command line, hasp_host_operators_console prompts the user for terminal input with the string "Input:".

Input from the terminal is transmitted directly to the remote system unless the line begins with the request character, an exclamation mark (!); lines beginning with the request character are interpreted by this command. The valid requests are described below.

Any text received from the remote system is displayed directly on the terminal without any interpretation by hasp_host_operators_console.

HASP_HOST_OPERATORS_CONSOLE REQUESTS

The following requests are recognized by hasp_host_operators_console when given at the beginning of a line of terminal input:

!.. <REST_OF_LINE>
: the rest of the line is passed to the Multics command processor for execution as ordinary commands.

!. : prints a message of the form:

hasp_host_operators_console N.N; connected to channel NAME.

where N.N is the current version of this program and NAME identifies the channel connected as a console to the remote system.

!quit : causes the command to hangup the operator's console channel and return to Multics command level.

Name: help

The help command prints the name of each command that can be executed by the driver. A short description of the arguments is provided with each command name. At request command level, the list of commands is limited to those unique to that command level.

Usage

help

Name: hold

The hold command holds the driver at command level.

Usage

hold

Notes

When the hold command is issued from the master terminal, the slave terminal is unable to issue any command that would cause the driver to leave command level until the master terminal has issued a go command (or a start command following a quit signal). This command should always be used following a quit signal if the automatic start is to be canceled.

Name: inactive_limit

The inactive_limit command allows the I/O Daemon to log out automatically after a specified period of inactivity.

Usage

inactive_limit {N}

where N is the number of minutes of inactivity allowed. N can be from zero to 200 minutes. Zero indicates no automatic logout; this is the default. The current inactivity limit is displayed if N is not given.

Notes

The inactivity time counter is reset when a request or command is received or a quit is signalled, as well as when the driver processes a new request. A driver sitting at command level is considered active.

An inactivity logout reinitializes a remote driver so that another station can log in and use the line.

Name: kill

The kill command terminates the request that the driver is currently processing. The request is passed back to the coordinator and placed in the saved list where it can be restarted if desired (within the limits of the coordinator save time).

After completing the command, the driver looks for another request to process. (In step mode, it returns to command level.)

Usage

kill

Name: logout

The logout command terminates the driver process (like the standard Multics logout command).

Usage

logout

Note

When the logout command is given from a remote station, the remote driver reinitializes and gets ready to accept a new station.

Name: new_device

The new_device command terminates the current device. The driver then asks the operator to enter a new "command or device/request_type."

The coordinator is notified of the termination of the current device and the device is detached by the process. If a control terminal has been attached, it also is detached.

The new_device command can only be issued from the master terminal.

Usage

new_device

Name: next

The next command specifies which request is to be taken from the queues next. This allows the operator to specify priority requests and the order in which they are to be run.

Usage

next -control_args

where the -user control argument is required and at least one other argument must be chosen from among the request identifiers (-entry, -path, and -id).

-user Person_id.Project_id
: specifies the submitter of the request by User_id. The full person and project names must be given.

-entry STR, -et STR
: specifies the entryname of the request. Starnames are not allowed. This control argument can not be used with the -path control argument.

-id ID
: specifies the match ID of the request.

-path path, -pn path
: specifies the full pathname of the request. Relative pathnames and starnames are not allowed. This control argument can not be used with the -entry control argument.

-device STR, -dev STR
: specifies which of the driver's minor devices the command is to affect. This control argument is optional for drivers with a single minor device, but is required for drivers with multiple minor devices. It serves to identify which request type the coordinator will search to find the request.

-queue N, -q N
: specifies that only queue N of the request type should be searched to find a matching request. This argument is optional; if not given, all queues are searched.

Note

All requests to be run by the next command are charged as though they came from queue 1.

Requests chosen to run next are run after any restarted requests (see the restart command in this section).

This command can be given several times before a go command, to specify the exact order that requests in the queues are processed.

Name: paper_info

The paper_info device-specific driver command defines the physical characteristics of the paper as used by the printer software.

Usage

paper_info {minor_device} {-control_args}

where minor_device is a minor device name (as shown by the status command) and is required if there is more than one printer minor device; control_args can be one or more of the following:

-print
: print the current values. If this is given, it must be the only control arg.

-ll N
: sets the line length to N, where N is a decimal integer from 10 to 200.

-pl N
: sets the page length to N, where N is a decimal integer from 10 to 127.

-lpi N
: sets the number of lines per inch to N, where N is either 6 or 8.

If no control arguments are given, the current values are printed.

Note

If the printer uses a firmware VFC image, a new image is loaded (which causes the printer to go into an unsynchronized state). Otherwise, the operator is told to mount a new VFU tape.

Name: pause_time

The pause_time device-specific driver command allows a remote device driver to accept commands between requests by pausing a few seconds to allow the line to turn around.

Usage

pause_time {N}

where N is the number of seconds that the driver must pause between requests. N must be between 0 and 30 seconds. If N is not given, a value of 10 is assumed.

Name: print

The print device-specific driver command starts the actual printing of a file when the driver is at request command level. This command is used by printer drivers only.

Usage

print {N}

where N is a decimal integer that identifies the page at which the driver starts printing. If this argument is omitted, printing starts at the current page of the file.

Notes

A "+" or "-" preceding the page number indicates that the number is relative to the current page.

If the starting page number is beyond the end of the file, an error message is printed, and a new command is requested.

The print command causes a normal head sheet to be printed complete with separator bars if needed. The head sheet is followed by the current page of the file.

Name: prt__control

The prt_control device-specific driver command sets the driver request processing modes. Each key can be preceded by the circumflex character (^) to set the value to off.

Usage

prt_control {minor_device} {args}

where minor_device is a minor device name (as shown by the status command) and is required if there is more than one printer minor device; args can be one or more of the following:

-print
: if arg is not given, or if a single arg "-print" is given, the current modes are printed.

auto_print, ^auto_print
: This mode causes the driver to start printing each request as soon as it is received from the coordinator (after a go command has been given). This is the normal mode of operation. When auto_print is turned off, the driver goes to request command level immediately after printing the log message. This allows the operator to align the paper, change the paper, print sample pages, and issue all other commands allowed at request command level (including the kill command).

force_esc, ^force_esc
: This mode turns on the esc mode of the printer DIM during the processing of each request. This mode must be on if the slew-to-channel functions are to operate. Normally, the force_esc mode is set by data in the request type info (rqti) segment.

force_nep, ^force_nep
: This mode sets the noendpage (nep) mode of the printer DIM during the processing of each request, whether the user has requested that mode or not. It is normally set from data in the rqti segment. This mode is used for request types that require preprinted or preformatted paper (e.g., gummed labels, invoice forms).

force_ctl_char
This sets the ctl_char mode of the printer DIM during the processing of each request, which allows an I/O daemon to send control sequences directly to a remote printer instead of discarding the characters or printing their octal equivalents. Setting this mode enables users who prepare print files through Compose to activate special printer features such as superscripting or multiple fonts. This mode is honored only by the remote printer driver module, remote_driver_.

If no arguments are given, the current modes are printed.

Name: punch

The punch command is used by remote punch drivers at request command level to proceed with the punching of the requested segment.

Usage

punch

Name: pun_control

The pun_control command is used by remote drivers at normal command level to set the punch control modes. This command does not apply to the central site punch driver.

Usage

pun_control {minor_device} {control_mode}

where:

1. minor_device
   is the name of the punch minor device that the command is addressing. This argument is optional if there in only one punch minor device, but is required otherwise.

2. control_mode
   specifies the modes to be set. The mode name can be preceded by the character "^" to reset the mode. This argument is optional. If not given, the current modes for the specified minor device are printed. The following mode is currently defined:

autopunch
this mode allows the driver to process punch requests continuously without operator intervention. When this mode is off (i.e., ^autopunch) the driver comes to request command level after printing the log message and waits for the operator to give the "punch" command before continuing.

-print
if control_mode is not given, or if a single argument "-print" is given, the modes are printed.

Notes

The ^autopunch mode is normally used by a remote operator to allow the output to be directed to a particular device based on information in the log message. Once the proper device has been assigned, the operator must type "punch" for the driver to continue with the user's request.

Name: read_cards

The read_cards device-specific driver command applies to device drivers that can read user card decks. It allows the operator to input card decks from a remote station or local device. The control card format required is the same as that described under "Reading User Card Decks" in the *Bulk I/O* manual.

Usage

read_cards

Notes

The card codes that are accepted by various card readers may vary from one card reader to another. The operator should be familiar with the card codes that should be used with the card reader at the remote station

---

Name: ready

The ready command places the device and the specified minor devices in the active or "ready" state. The driver only requests service from the coordinator for a ready device. This command performs the reverse function of the halt command.

Usage

ready dev1 ... dev_ {-control_arg}

where:

1. dev_
   is the name of a device, or minor device in the case of a multifunction device, that is to be placed in the ready state.

2. -control_arg
   can be -all or -a to place all devices in the ready state. If the -all control argument is used, no device names need be given. If there is only one device, no control argument is required. In this case, the ready command is executed automatically during driver initialization. If there are multiple minor devices, the operator must specify the ones to be made ready or else must specify -all to make all minor devices ready.

Name: receive

The receive command causes the driver to wait for output files to be transmitted from the remote system. A message is issued at the start and end of each file received. If automatic queueing of output files is enabled for this simulated device, output files are locally printed or punched after they have been successfully received; otherwise, the output files are placed into system pool storage as specified by the ++IDENT control records that must be present in the files.

After use of the receive command, the driver only recognizes pending commands while it is between output files. If it is necessary to execute a command while a file is being received, a QUIT must be issued to the driver to bring the driver to QUIT command level. The hold command can then cause the driver to remain at QUIT level; the release command can abort receiving the file and return to normal command level; and the start command can resume receiving the file.

Usage

receive

Name: reinit

The reinit command reinitializes the driver. The same device(s) and request type(s) are used without requesting operator input. However, remote stations have to reissue the station command and any new default request types. Also, if a control terminal is attached to the driver, its attachment, form simulation mode, and form type are retained over the reinitialization. Each device and request type is again requested from the coordinator.

The reinit command to the driver is almost the same as the standard Multics new_proc command.

Usage

reinit

Name: release

The release command returns the driver to normal command level. This command is primarily used following a quit signal. If a request was in progress, it is started over again.

Usage

release

Name: req_status

The req_status device-specific driver command gives the operator information about the current request. This command can only be used at request command level.

Usage

req_status {-control_arg}

where control_arg, for printers only, can be -long or -lg to give the operator the following information:

number of multisegment file components
number of characters in file
current page number
current copy number
current line count
current multisegment file component
char offset in current component
char offset from start of file
printer DIM modes
printer DIM position

If the control argument is omitted, only the first four items in the above list are printed. In this case, the information looks like:

```
Request 10001: >print_files>invoices>Station_A.invoices
   file components:  2,    char count:  4732865
   page no:          1006     current copy no: 2
```

There is no control_arg defined for punches. The following three items are printed:

current copy number
current request number
current pathname

In this case, the information looks like:

```
Request 20001 >punch_files>invoices>Station_A.invoices
   current copy no:  2
```

Name: request_type, rqt

The request_type command is used to specify the request type to be used for the automatic queuing of output files received by this device.

Usage

rqt rqt_name

where rqt_name is the name of the request type to be used for automatic queuing. The generic type of this request type must agree with the type of device being simulated (e.g., "printer" for simulated line printers). This parameter is optional; the default value is the request type specified in the iod_tables definition of this driver.

Name: restart

The restart command is used either to restart processing of the current request after a device malfunction or to reprocess requests in the coordinator's saved list.

Usage

restart {arg}

where arg can be one of the following:

1. N

   is the number of the request to be restarted. The coordinator searches its saved list for a matching request. If found, the request is reprocessed ahead of any other requests, including those from the "next" command. If the request had been saved in the middle of a copy (suspended), the request is restarted beginning at the top of the following page; a punch request starts at the beginning of that copy.

2. -from N

   specifies that all requests in the series beginning with request N are to be restarted. This is an implicit save of all requests in the series.

When the restart command is issued directly after a quit signal, with no arguments, the driver's current request is restarted. For print requests, the current page number, minus 5, and copy number are displayed and the driver goes to request command level. For punch requests, the number of copies completed (if more than one) is displayed and the operator is asked to note how many were good.

Notes

The user is charged for the requested number of copies only, regardless of how many copies were produced by this command.

If the request number series of a restarted request is still active, the driver is switched to another series. Each restarted request is assigned a new request number, and any subsequent restart must be based on the new request number.

Name: runout_spacing

The runout_spacing device-specific driver command sets the number of lines to advance the paper after requesting a command from a remote multifunction slave terminal.

Usage

runout_spacing N

where N is the number of lines the driver advances the paper after requesting a command from the slave. N can be from zero to 60.

Note

The runout spacing is normally set in the attach description from the iod_tables. This command allows the operator to change the spacing so that driver command requests can be seen clearly above the platen.

Name: sample

The sample device–specific driver command is used by printer drivers at request command level to print a sample page of the file for paper alignment or to verify the starting position in the file. The current position of a new request is always page 1. The same page can be printed as often as needed.

Usage

sample {N}

where N is the page number that the driver prints. If N is omitted, the driver prints the current page in the file.

If N is preceded by a "+" or "–", the number is relative to the current page of the file. For example, "sample +3" skips forward three pages and prints the page; "sample –8" skips backward eight pages and prints the page. Similarly, "sample 500" skips to page number 500 and prints it.

If the page number specified is beyond the end of the file, an error message is printed similar to:

```
End–of–File record encountered. EOF at page 2000, line 10.
Unable to skip to starting page.
Enter command(request):
```

and a new command is requested.

The sample command prints a page with separator bars as an aid to the operator in indicating the sample pages so they can be discarded.

Name: sample_form

The sample_form device–specific driver command prints a sample of the data used to record request processing on the control terminal. The primary function of this command is to verify the alignment of the forms on the control terminal. The data is formatted by the program that is called for each copy of each request. (See the ctl_term command.)

Usage

sample_form

Notes

If form feed simulation is being used, the command checks to see if alignment has been set. If not, it is set before the sample form is printed.

The sample_form command applies to all drivers that use a control terminal.

Name: sample_hs

The sample_hs device-specific driver command prints a sample head sheet to align the paper before starting to print or after loading more paper. This command should not be used in the middle of a request (e.g., after a quit) unless the request is restarted using the restart command. Otherwise, the page restart feature of the printer driver is placed out of synchronization.

Usage

sample_hs {minor_device}

where minor_device is a minor device name (as shown by the status command) and is required if there is more than one printer minor device.

Name: save

The save command tells the coordinator that one or a series of requests are to be retained beyond the normal holding time. The action is limited to requests in the specified request number series. The save command allows requests to be saved for possible restarting until the coordinator is logged out.

Usage

save {arg}

where arg can be one of the following:

1. N

   specifies the request number in the coordinator's saved list. The coordinator searches its list of finished requests and marks the matching request number as saved for later restarting. The request remains in the saved list until the request is restarted by the restart command or until the coordinator is next initialized.

2. -from N

   specifies that all requests in the series beginning with request N are retained in the saved list.

If no argument is given, the current request is returned to the coordinator and saved for later restarting. For printers, the request is processed to the bottom of the next even page and a normal tail sheet is printed, showing a charge of zero. When the request is later restarted, printing begins at the top of the next odd page.

Notes

Once a saved request is restarted, it is not saved any longer than the normal retention time. The coordinator never deletes the user's segment while the request is being saved.

Name: sep_cards

The sep_cards command is used by a remote punch driver at normal command level to control the punching of separator cards between each output deck. If separator cards are not punched, the operator should run the driver in step mode (see the step mode command) and remove the cards from the punch as each request is completed.

Usage

sep_card {minor_device} {arg}

where:

1. minor_device

   is the name of the punch minor device being addressed. This argument is optional if there is only one punch minor device, but is required otherwise.

2. arg

   can be one of the following:

   standard
   the standard separator cards are to be punched (default).

none
no separator cards are to be punched.

-print
if arg is not given, or if a single arg "-print", is given, the current value is printed.

Name: single

The single device-specific driver command applies only to drivers that operate a printer. It sets the single mode of the printer DIM so that formfeed and vertical tab characters are treated as newline characters for the current request. It also cancels any additional requested copies that have not been processed by the driver. The single command is used after a quit to stop runaway paper feeding caused, for example, by the printing of a non-ASCII segment.

Usage

single

Name: start

The start command allows the driver to resume operations suspended at other than the normal command level, e.g., after a quit signal. Its function is similar to the standard Multics start command. The start command cannot be issued at normal command level (see the go command).

After a quit signal, this is the only command that allows control to be returned to the point of process interruption. The action of the hold command is reset when a start command is issued.

Usage

start

Name: station

The station command is used by a driver to identify and validate a remote station. This command is similar to the standard Multics login command.

Usage

station station_id {station_password}

where:

1. station_id
   is the registered id of the station, as defined by the administrator.

2. station_password
   is the registered password for the remote station.

Notes

The station's identifier and password are registered in the PNT using the card input password as the station password and are supplied by the administrator for each station location.

If the remote station includes an operator's terminal with keyboard and CRT or printer, the station password can be omitted from the station command. The system then requests the station password and either suppresses printing of the password or masks it. This feature is particularly useful when a remote station is actually a high-quality letter printer (e.g., a Diablo 1640), where the printer is used both as the slave console and as the actual output device.

Remote stations that have no input device do not have to give a station command. However, these stations must use a dedicated phone line and have the station identifier specified in the iod_tables as described earlier for Type II remote stations.

Name: status

The status command prints information about the current status of the driver. The information provided is:

1. The I/O daemon driver version.
2. The device name and channel.
3. The request type (per minor device if more than one).
4. Whether a request is in progress and the request number.
5. The device status: ready, halted, or not attached. (If there are minor devices, this is provided per minor device.)
6. Pending requests and their request numbers.
7. Whether step mode is set.
8. The names of any minor devices (to be used with the ready and commands).

Usage

status {-control_arg}

where control_arg can be -long or -lg to print the status of inactive minor devices (devices that cannot be made ready).

Name: step

The step command either sets (puts the driver into) or resets (takes the driver out of) step mode. When in step mode, the driver returns to command level after processing each request from the coordinator. When not in step mode, the driver processes requests from the coordinator as soon as received without operator interaction. Step mode is useful for checking the alignment of paper on the printer or other device functions prior to allowing the driver to run continuously without operator interaction.

Usage

step {arg}

where arg can be "set" or "reset" to put the driver into or take the driver out of step mode. If no argument is supplied, step mode is set. The driver is not in step mode immediately after driver initialization.

Name: x

The x command allows drivers to execute an admin exec_com on a site-defined basis.

Usage

x function {args}

where:

1. function
   is a site-defined function name.

2. args
   are any arguments needed to implement function.

Notes

When the user issues the x command, the driver constructs the command line:

exec_com >ddd>idd>NAME function {args}

where function and args are as above; NAME is either <major_device>_admin.ec for standard drivers or <station_id>_admin.ec for remote drivers. If NAME is not found, the driver looks for the default of iod_admin.ec. Added names can be used to group exec_coms into categories.

Drivers that run as IO.SysDaemon have a great deal of access to the storage system. Administrators must be careful in choosing commands for the admin exec_coms to avoid accidents or vandalism.

The Multics iod_command command can be used within an admin exec_com to execute arbitrary I/O daemon commands. For example:

iod_command defer_time 30

in an admin exec_com changes the auto defer time limit for the current driver to 30 minutes. The iod_command command is described in the *Bulk I/O* manual.

SECTION 6

I/O MODULES

Name: hasp_host_

The hasp_host_ I/O module simulates record-oriented I/O to a single device of a workstation while communicating with a host system using the HASP communications protocol. See the "Notes" below for more detail.

Entry points in this module are not called directly by users; rather, the module is accessed through the I/O system.

This I/O module must be attached to a subchannel of a communications channel configured to use the HASP ring-0 multiplexer.

This I/O module is designed primarily for use by the Multics I/O daemon.

Attach Description

hasp_host_ -control_args

where control arguments can be chosen from the following and are optional, with the exception of -comm, -tty, and -device:

-comm hasp
: is required for compatibility with other I/O modules used by the I/O daemon.

-tty channel_name
: specifies the communications channel to be attached. The channel must be a subchannel of a HASP multiplexed channel (e.g., a.h014.prt3).

-device STR
: specifies the type of device for this attachment. STR must be one of "teleprinter", "reader", "printer", or "punch". The type specified by this control argument must match the type of device attached to the channel name defined above.

-terminal_type STR, -ttp STR
is optional and is used to define the character set used by the remote system. STR must be the name of a terminal type defined in the site's Terminal Type Table (TTT). See the section "Character Set Specification" below for more information, including the default character set used if this control argument is omitted.

-physical_line_length N, -pll N
is accepted for compatibility with other I/O modules used by the I/O daemon, but is ignored by this I/O module.

-ebcdic
is accepted for compatibility with other I/O modules used by the I/O daemon, but is ignored by this I/O module.

Open Operation

The hasp_host_ I/O module supports the sequential_input, sequential_output, and sequential_input_output opening modes.

Write Record Operation

The write_record entry converts the supplied data record from ASCII to the remote system's character set, performs data compression, and transmits the record to the HASP multiplexer.

The format of the record supplied to this I/O module follows. This structure and the referenced constants are contained in the terminal_io_record.incl.pl1 include file:

```
dcl 1 terminal_io_record aligned based,
    2 version fixed binary,
    2 device_type fixed binary,
    2 slew_control,
      3 slew_type fixed binary (18) unaligned unsigned,
      3 slew_count fixed binary (18) unaligned unsigned,
    2 flags,
      3 binary bit (1) unaligned,
      3 preslew bit (1) unaligned,
      3 pad bit (34) unaligned,
    2 element_size fixed binary,
    2 n_elements fixed binary (24),
    2 data,
      3 bits (terminal_io_record_n_elements refer
                  (terminal_io_record.n_elements))
        bit (terminal_io_record_element_size refer
                  (terminal_io_record.element_size)) unaligned;
```

where:

version (Input)
is the current version of this structure. This version of the structure is given by the value of the named constant terminal_io_record_version_1.

device_type (Input)
is the type of device to which this record is to be written. The acceptable values are TELEPRINTER_DEVICE and READER_DEVICE.

slew_control (Input)
is ignored by this I/O module as the HASP communications protocol does not define slew operations for either the teleprinter or card reader.

flags.binary (Input)
must be set to "0"b. (This I/O module does not support binary data transmission.)

flags.preslew (Input)
must be set to "0"b.

element_size (Input)
must be set to 9. (This I/O module only supports transmission of characters.)

n_elements (Input)
is the number of characters in the record to be written.

data.bits (Input)
is the actual data. This I/O module expects to be supplied ASCII characters.

Read Record Operation

The read_record entry returns a single record from the device, basically performing the inverse of the functions described for the write_record operation. Additionally, for line printer attachments, the carriage control information in the record is converted into the appropriate slew information in the terminal_io_record.

The format of the record that this I/O module returns in the supplied buffer is as follows. The structure and the referenced constants are contained in the terminal_io_record include file:

```
dcl 1 terminal_io_record aligned based,
    2 version fixed binary,
    2 device_type fixed binary,
    2 slew_control,
      3 slew_type fixed binary (18) unaligned unsigned,
```

```
      3 slew_count fixed binary (18) unaligned unsigned,
    2 flags,
      3 binary bit (1) unaligned,
      3 preslew bit (1) unaligned,
      3 pad bit (34) unaligned,
    2 element_size fixed binary,
    2 n_elements fixed binary (24),
    2 data,
      3 bits (terminal_io_record_n_elements refer
                (terminal_io_record.n_elements))
        bit (terminal_io_record_element_size refer
                (terminal_io_record.element_size)) unaligned;
```

where:

version (Output)
is the current version of this structure. This version of the structure is given by the value of the named constant terminal_io_record_version_1.

device_type (Output)
is the type of device from which this record was be read. Its possible values are TELEPRINTER_DEVICE, PRINTER_DEVICE, or PUNCH_DEVICE.

slew_control (Output)
if the input device is a line printer, this substructure is filled in with the interpretation of the HASP carriage control record present in each line printer record; otherwise, it is always set to the value specified below.

slew_type (Output)
for a line printer, is set to the type of slew operation to be performed before/after "printing" the data in the record and may be either SLEW_BY_COUNT or SLEW_TO_CHANNEL. For a teleprinter or punch it is set to SLEW_BY_COUNT. (The data returned is processed by the caller of this I/O module; this processing is herein termed the "printing" of the data.)

slew_count (Output)
for a line printer, is set to the value to be interpreted according to slew_control.slew_type above. For a teleprinter or punch it is set to 1.

flags.binary (Output)
is always set to "0"b.

flags.preslew (Output)
for a line printer, is set to "1"b if the slew operation above is to be performed before "printing" the data in the record, or is set to "0"b if the slew operation is to be performed after "printing". For other than the line printer, it is always set to "0"b.

element_size (Output)
is always set to 9.

n_elements (Output)
is set to the number of characters returned in the record.

data.bits (Output)
is the actual returned data. This I/O module converts the data input from the remote host to ASCII.

Control Operation

This I/O module supports the following control operations:

runout
ensures that all data has been transmitted to the HASP multiplexer from where it is guaranteed to be transmitted to the terminal.

end_write_mode
ensures that all previously written data has been transmitted to the HASP multiplexer and then writes an end-of-file record for the device.

read_status
determines whether or not there are any records waiting for a process to read. The info_ptr should point to the following structure, which is filled in by the call:

```
dcl 1 info_structure aligned,
      2 ev_chan fixed bin (71),
      2 input_available bit (1);
```

where:

ev_chan (Output)
is the event channel used to signal the arrival of input.

input_available (Output)
indicates whether input is available:
"0"b no input
"1"b input

resetread
discards any pending input.

resetwrite
discards any unprocessed output.

hangup_proc
is used to specify a procedure to be invoked when this attachment's channel is hung up. The info_ptr points to the following structure:

```
dcl 1 hangup_proc_info aligned,
    2 procedure entry variable,
    2 data_ptr pointer,
    2 priority fixed binary;
```

where:

procedure (Input)
is the procedure to be invoked when the hangup occurs.

data_ptr (Input)
is a pointer to be supplied to the procedure.

priority (Input)
is the priority for the hangup event.

A detailed explanation of data_ptr and priority can be found in the description of ipc_ in the *Multics Subroutines* manual.

select_device
reset
are ignored rather than rejected for compatibility with other I/O modules used by the I/O daemon.

signon_record
no_signon_record
can only be issued on the operator's console subchannel of the multiplexer. These are described in the "SIGNON Processing" section.

Modes Operation

This module accepts the "non_edited" and "default" modes for compatibility with other I/O modules used by the I/O daemon, but ignores them.

Character Set Specification

This I/O module allows the specification of the character set used by the remote system through the -terminal_type attach option.

If -terminal_type is given, the referenced terminal type must be defined in the site's TTT with both an input and output translation table. This module uses these translation tables to convert data from the remote system to ASCII, and from ASCII to the remote system's character set.

If -terminal_type is not given, the remote system is assumed to use EBCDIC as its character set. In this case, the ascii_to_ebcdic_ subroutine converts data sent to the system; the ebcdic_to_ascii_ subroutine converts data received from the remote system. (See *Multics Subroutine* manual for a description of these translations.)

SIGNON Processing

Before communicating with certain remote systems, Multics must send the SIGNON record. This specially formatted record identifies Multics to the remote system.

For these systems, the Multics multiplexer must be configured to use "signon_mode". Before data transmission is permitted, the signon_record control order must be issued on an I/O switch attached to the operator's console subchannel of the multiplexer.

If the remote system does not expect a SIGNON record, the "no_signon_record" control order can be used to validate that the multiplexer channel is properly configured.

sign_on_record Control Order

This control order supplies a SIGNON record for transmission to the remote system. The info_ptr must locate the following structure declared in the hasp_signon_record_info.incl.pl1 include file:

```
dcl 1 signon_record_info aligned based,
    2 version fixed binary,
    2 pad  bit (36),
    2 event_channel fixed binary (71),
    2 record character (80) unaligned;
```

where:

version
: is the current version of this structure. It must have the value of the named constant SIGNON_RECORD_INFO_VERSION_1.

pad
: is reserved for future expansion and must be zero.

event_channel
: is an event-wait channel whose use is described below.

record
: is the actual text of the SIGNON record in ASCII. This I/O module translates the text to uppercase and the remote system's character set.

If the status code returned by this control order is zero, the calling program must block on the above event-wait channel. When the wakeup arrives, the event message indicates the success or failure of the control order. It has one of the following values (found in the named include file):

HASP_SIGNON_OK
indicates that the remote system has accepted the SIGNON record.

HASP_SIGNON_REJECTED
indicates that the remote system has rejected the record; the caller should try again with a different record.

HASP_SIGNON_HANGUP
indicates that the remote system has rejected the record and disconnected the multiplexer.

If the status code returned by the control order is error_table_$invalid_state, the multiplexer is not configured to send a SIGNON record.

no_signon_record Control Order

This control order validates that the multiplexer is not configured to send a SIGNON record to the remote system. This order does not accept an info structure.

If the returned status code is error_table_$invalid_state, the multiplexer is configured to send a SIGNON record, and a "signon_record" must be issued on this subchannel.

get_device_type Control Order

This control order is recognized for HASP subchannels in addition to the control orders defined in the description of the tty_ I/O module in the *Multics Subroutines* manual. It is used exclusively by the hasp_workstation_ and hasp_host_ I/O modules and should not be invoked by user programs.

In the following description, the "info_ptr" field provides the declaration of the item that the caller must supply to the HASP multiplexer software.

The get_device_type control order returns the type of device (operator's console, card reader, line printer, or card punch) connected to the subchannel.

info_ptr: fixed binary (35) (Output)
return codes:
0 the device type is supplied in the given field
error_table_$undefined_order_request
this channel is not part of a HASP multiplexer.

The possible values returned by this control order are defined in the hasp_device_data_incl.pl1 include file.

Notes

As stated above, this I/O module is used to simulate the operation of a single device of a HASP workstation.

If the simulated device is a card reader, the caller supplies records to this module that are then formatted and transmitted to the remote host. In other words, a card reader attachment through this switch is an output-only attachment.

Similarly, this I/O module receives records from the remote host when the simulated device is either a line printer or card punch. Thus, line printers and card punches attached through this I/O module are input-only devices.

Special I/O daemon software is provided to allow Multics to simulate the operations of a workstation in order to submit jobs to remote systems and receive those jobs' output print and punch files. This workstation simulator uses this I/O module for communications with the remote host.

hasp_workstation_

Name: hasp_workstation_

The hasp_workstation_ I/O module performs record-oriented I/O to a single device of a remote terminal that supports the HASP communications protocol.

Entry points in this module are not called directly by users; rather, the module is accessed through the I/O system.

This module must be attached to a subchannel of a communications channel configured to use the HASP ring-0 multiplexer.

The module is designed primarily for use by the Multics I/O daemon. It expects output for the operator's console and line printers to have been properly formatted by the prt_conv_ module.

Attach Description

hasp_workstation_ -control_args

where control arguments can be chosen from the following and are optional, with the exception of -comm, -tty, and -device:

-comm hasp
: is required for compatibility with other I/O modules used by the I/O daemon.

-tty channel_name
: specifies the communications channel to be attached. The channel must be a subchannel of a HASP multiplexed channel (e.g., a.h014.prt3).

-device STR
: specifies the type of device for this attachment. STR must be one of "teleprinter", "reader", "printer", or "punch". The type specified by this control argument must match the type of device attached to the channel name defined above.

-terminal_type STR, -ttp STR
: is optional and is used to define the character set used by the remote terminal. STR must be the name of a terminal type defined in the site's Terminal Type Table (TTT). See the section "Character Set Specification" below for more information, including the default character set used if this control argument is omitted.

-physical_line_length N, -pll N
: is accepted for compatibility with other I/O modules used by the I/O daemon, but is ignored by this I/O module.

-ebcdic
is accepted for compatibility with other I/O modules used by the I/O daemon, but is ignored by this I/O module.

-top_of_page STR
specifies the sequence of carriage control operations to be used to move to the top of the next page. This control argument is only permitted for a line printer. The format of STR is described in "Carriage Control Specifications" below. (Default is "c1".)

-inside_page STR
specifies the sequence of carriage control operations to be used to move to the top of the next inside page. An inside page is the page on which the I/O daemon prints head sheets. This control argument is only permitted for a line printer. The format of STR is described in "Carriage Control Specifications" below. (Default is "c1".)

-outside_page STR
specifies the sequence of carriage control operations to be used to move to the top of the next outside page. An outside page is the page on which the I/O daemon prints tail sheets. This control argument is only permitted for a line printer. The format of STR is described in "Carriage Control Specifications" below. (Default is "c1".)

-forms STR
specifies the type of forms to be used to print output directed through this attachment. STR is an arbitrary string of, at most, 32 characters whose interpretation is site dependent. This control argument is only permitted for a line printer. (Default is the null string.)

Open Operation

The hasp_workstation_ I/O module supports the sequential_input, sequential_output, and sequential_input_output opening modes.

Write Record Operation

The write_record entry converts the supplied data record from ASCII to the remote terminal's character set, converts the supplied slew control into the proper carriage control sequences for line printer attachments, performs data compression, and transmits the record to the HASP multiplexer.

The format of the record supplied to this I/O module follows. This structure and the referenced constants are contained in the terminal_io_record include file:

```
dcl 1 terminal_io_record aligned based,
    2 version fixed binary,
    2 device_type fixed binary,
    2 slew_control,
```

```
        3 slew_type fixed binary (18) unaligned unsigned,
        3 slew_count fixed binary (18) unaligned unsigned,
      2 flags,
        3 binary bit (1) unaligned,
        3 preslew bit (1) unaligned,
        3 pad bit (34) unaligned,
      2 element_size fixed binary,
      2 n_elements fixed binary (24),
      2 data,
        3 bits (terminal_io_record_n_elements refer
                (terminal_io_record.n_elements))
          bit (terminal_io_record_element_size refer
                (terminal_io_record.element_size)) unaligned;
```

where

version (Input)
is the current version of this structure. This version of the structure is given by the value of the named constant terminal_io_record_version_1.

device_type (Input)
is the type of device to which this record is to be written. The acceptable values are TELEPRINTER_DEVICE, PRINTER_DEVICE, or PUNCH_DEVICE.

slew_control (Input)
need only be supplied by the caller if device_type is PRINTER_DEVICE and specifies the slew operation to be performed after printing the data in the record.

slew_type (Input)
specifies the type of slew operation. The possible values are SLEW_BY_COUNT, SLEW_TO_TOP_OF_PAGE, SLEW_TO_INSIDE_PAGE, SLEW_TO_OUTSIDE_PAGE, or SLEW_TO_CHANNEL.

slew_count (Input)
is interpreted according to the value of slew_control.slew_type.

flags.binary (Input)
must be set to "0"b. (This I/O module does not support binary data transmission.)

flags.preslew (Input)
must be set to "0"b. (This I/O module does not support slew operations before printing the record's data.)

element_size (Input)
must be set to 9. (This I/O module only supports transmission of characters.)

hasp_workstation_

n_elements (Input)
is the number of characters in the record to be written.

data.bits (Input)
is the actual data. This I/O module expects to be supplied ASCII characters.

Read Record Operation

The read_record entry returns a single record from the device, basically performing the inverse of the functions described for the write_record operation.

The format of the record this I/O module returns in the supplied buffer follows. This structure and the referenced constants are contained in the terminal_io_record include file:

```
dcl 1 terminal_io_record aligned based,
    2 version fixed binary,
    2 device_type fixed binary,
    2 slew_control,
      3 slew_type fixed binary (18) unaligned unsigned,
      3 slew_count fixed binary (18) unaligned unsigned,
    2 flags,
      3 binary bit (1) unaligned,
      3 preslew bit (1) unaligned,
      3 pad bit (34) unaligned,
    2 element_size fixed binary,
    2 n_elements fixed binary (24),
    2 data,
      3 bits (terminal_io_record_n_elements refer
                (terminal_io_record.n_elements))
        bit (terminal_io_record_element_size refer
                (terminal_io_record.element_size)) unaligned;
```

where:

version (Output)
is the current version of this structure. This version of the structure is given by the value of the named constant terminal_io_record_version_1.

device_type (Output)
is the type of device from which this record was read. Its possible values are TELEPRINTER_DEVICE or READER_DEVICE.

slew_control.slew_type (Output)
is always set to SLEW_BY_COUNT.

slew_control.slew_count (Output)
is always set to 1.

hasp_workstation_

flags.binary (Output)
is always set to "0"b.

flags.preslew (Output)
is always set to "0"b.

element_size (Output)
is always set to 9.

n_elements (Output)
is set to the number of characters returned in the record.

data.bits (Output)
is the actual returned data. This I/O module converts the data input from the remote workstation to ASCII.

Control Operation

This I/O module supports the following control operations:

runout
ensures that all data has been transmitted to the HASP multiplexer from where it is guaranteed to be transmitted to the terminal.

end_write_mode
ensures that all previously written data has been transmitted to the HASP multiplexer and then writes an end-of-file record for the device.

read_status
determines whether or not there are any records waiting for a process to read. The info_ptr should point to the following structure, which is filled in by the call:

```
dcl 1 info_structure aligned,
    2 ev_chan fixed bin (71),
    2 input_available bit (1);
```

where:

ev_chan (Output)
is the event channel used to signal the arrival of input.

input_available (Output)
indicates whether input is available:
"0"b no input
"1"b input

resetread
flushes any pending input.

hasp_workstation_

resetwrite
flushes any unprocessed output.

hangup_proc
is used to specify a procedure to be invoked when this attachment's channel is hung up. The info_ptr points to the following structure:

```
dcl 1 hangup_proc_info aligned,
    2 procedure entry variable,
    2 data_ptr pointer,
    2 priority fixed binary;
```

where:

procedure (Input)
is the procedure to be invoked when the hangup occurs.

data_ptr (Input)
is a pointer to be supplied to the procedure.

priority (Input)
is the priority for the hangup event.

A detailed explanation of data_ptr and priority can be found in the description of ipc_ in the *Multics Subroutines* manual.

select_device
reset
are ignored rather than rejected for compatibility with other I/O modules used by the I/O daemon.

Modes Operation

This module accepts the "non_edited" and "default" modes for compatibility with other I/O modules used by the I/O daemon, but ignores them.

Character Set Specification

This I/O module allows the specification of the character set used by the remote workstation through the -terminal_type attach option.

If -terminal_type is given, the referenced terminal type must be defined in the site's TTT with both an input and output translation table. This module will use these translation tables to convert data from or to the remote workstation from or to ASCII, respectively.

hasp_workstation_

If -terminal_type is not given, the remote system is assumed to use EBCDIC as its character set. In this case, the subroutine ascii_to_ebcdic_ is used to convert data sent to the workstation; the subroutine ebcdic_to_ascii_ is used to convert data received from the remote system. (See *Multics Subroutines* manual for a description of these translations.)

Carriage Control Specifications

Multics I/O daemon software uses three special slew operations -- skip to top of the next page, skip to top of the next inside page, and skip to the top of the next outside page. (An inside page is the type of page on which the I/O daemon would print a head sheet; an outside page is the type on which it would print a tail sheet.)

By default, this I/O module assumes that all of these slew operations can be simulated on the remote workstation's line printer by skipping to channel one. However, through use of the -top_of_page, -inside_page, and -outside_page control arguments, any sequence of carriage motions can be specified to simulate these slew operations.

The format of this carriage control specification is:

Tn:Tn:Tn:...

where "n" is a numeric value and "T" represents how to interpret that numeric value. "T" can be either "c", representing skip to channel "n", or "s", representing slew "n" lines.

For example, the string:

c7:s5:c12

means skip to channel seven, space five lines, and finally skip to channel twelve.

get_device_type Control Order

This control order is recognized for HASP subchannels in addition to the control orders defined in the description of the tty_ I/O module in the *Multics Subroutines* manual. It is used exclusively by the hasp_workstation_ and hasp_host_ I/O modules and should not be invoked by user programs.

In the following description, the "info_ptr" field provides the declaration of the item that the caller must supply to the HASP multiplexer software.

The get_device_type control order returns the type of device (operator's console, card reader, line printer, or card punch) connected to the subchannel.

info_ptr: fixed binary (35) (Output)
return codes:
0 the device type is supplied in the given field
error_table_$undefined_order_request
this channel is not part of a HASP multiplexer.

The possible values returned by this control order are defined in the hasp_device_data_incl.pl1 include file.

SECTION 7

TEST MODE

SETTING UP THE TEST DIRECTORY

The test environment allows you to test the software and data base changes you made to get your HASP system working. Set up a test directory with structure similar to the >ddd>idd directory, giving test users sma access. Create an rqt_info_segs directory there, too, containing all rqti segments you plan to use. If card input is to be performed, create a card_pool directory in the test directory as well, assigning it sufficient quota for the card input.

Create the iod_tables.iodt segment in the test directory, compiled via the iod_tables_compiler command. If the driver is run from other than an IO.SysDaemon process (e.g., the process attaching the simulated operator's console), each request type used requires the following in iod_tables.iodt:

```
driver_userid:          Person_id.Project_id;
accounting:             nothing;
```

where Person_id.Project_id identifies the testing process. You must also create message segment queues for the request types to be used during testing (via create_daemon_queues command or the message segment commands).

Check that the testing process is using the TTT in which you have defined the HASP multiplexer terminal type; that it has the dialok attribute in the PDT, and rw access to the access control segments for the communications channels and card input devices.

If the x command is to be used during testing, include an iod admin exec_com or device admin exec_com in the test directory.

MANIPULATING REQUESTS IN THE TEST QUEUES

Since the test driver process will be using message segments in the test directory, the dprint, dpunch, list_daemon_requests (ldr) and cancel_daemon_requests (cdr) commands must be made aware of the test environment. This is done by calling special entries in each command procedure and indicating the test directory (TEST_DIR in examples) as follows:

```
dprint_$test TEST_DIR
ldr$test_ldr TEST_DIR
cdr$test_cdr TEST_DIR
```

Once this is done, the normal system printer/punch queues are no longer known to the test process.

THE TEST PROCESS

A standard I/O daemon process operates either as a coordinator or as a driver, with only one coordinator operating on the system at any time. In test mode, a single test process can perform the functions of both coordinator and driver; or, after one interactive test process has become a coordinator, another interactive process can become a driver. The second interactive process must use the same test directory as the first process. The test processes acting as coordinator and driver are unknown to the standard system I/O daemon processes.

Experimental software should exist in either bound or loose form in the test directory. If one component of a bound object segment is loose, then all components must be loose. You may want to initiate each object segment first.

Start the test process by calling the test entry of the iod_overseer_ subroutine:

```
test_io_daemon -dr TEST_DIR
```

When running the coordinator and driver in a single test process, the dialog from this point looks like the following, with user responses preceded by an exclamation point (!):

```
  Enter command:  coordinator or driver
! coord
  I/O Coordinator Version: X.X
  I/O Coordinator initialized
! driver

  I/O Daemon Driver Version: X.X
  Driver running in test mode.


  Enter command or device/request type:
```

At this point the driver will accept a device name to run a printer, punch, or remote workstation device.

Because the test entry was used, several commands are available to the user. One of these is the debug command, which calls the system debug command. From within the debug command, the user can use all the the debug command requests, including ".." to execute normal Multics commands.

Within the coordinator/driver test process there exist two pseudo processes stacked above the original interactive process: the coordinator in the middle, and the driver on top. Your console terminal communicates with the driver process after you have typed in "driver" during initialization. If you issue the logout command, you log out only the driver part of the test process; the console is then communicating with the coordinator

part of the test process. You can now start a new driver servicing the same or another device defined in the test directory's iod_tables. To terminate the test session, issue the logout command again, and the coordinator part of the process logs out. You are now back to normal Multics interactive command level.

Setting Breakpoints

You may wish to set breaks in the software to investigate a problem. Create and initiate a copy of the desired segment in the test directory. If the segment normally exists in a bound object segment, all components must exist and be initiated in the test directory. At your option, the source can be copied into the test directory and recompiled with the map and table options. This allows full use of either the probe or the debug command to investigate the problem.

To set breaks with the debug command, enter debug, set the breaks, and then bring up the test driver from within debug. This way the process transfers directly to debug whenever a breakpoint is reached.

To set breaks with the probe command, enter probe, set the breaks, and optionally bring up the test driver within probe. If the test driver is already initialized, the debug command must be given in order to enter probe (via the debug request "..probe") to manipulate breaks previously set up by probe, unless the process is stopped at a probe break.

Some errors occurring before full driver initialization invoke debug automatically while in test mode. The state of the process can be examined at this point. A ".q" debug request performs the equivalent of a start command.

Test mode command level is indicated parenthetically in the command level message as:

Enter command (iodd signal):

SAMPLE EXEC_COM FILE

The following is a sample of an exec_com that has proven useful in setting up and running a test environment. When creating your own exec_com, remember to replace TEST_DIR with the absolute pathname of the test directory.

```
&command_line off
&goto &ec_name

&label setup_environment
sa TEST_DIR>** sma [user name].[user project]
sa TEST_DIR>coord_dir>** rw [user name].[user project]
sa TEST_DIR>coord_lock rw
sa TEST_DIR>iodc_data rw
```

```
mssa TEST_DIR>([segs *.ms]) adros [user name].[user project]
& Initiate software in test directory at this point.
& set_ttt_path TEST_DIR>TTF.ttt
&quit

&label start_iod
&attach
test_io_daemon -dr TEST_DIR
coord
driver
&detach
&quit

&label use_test_queues
& Call the test entry of the daemon request commands.
dprint_$test TEST_DIR
ldr$test_ldr TEST_DIR
cdr$test_cdr TEST_DIR
&quit

&label use_system_queues
dprint_$test >ddd>idd
ldr$test_ldr >ddd>idd
cdr$test_cdr >ddd>idd
&quit

&label make_tables
& Compile the iod_tables and generate any missing message segments.
iodtc iod_tables
create_daemon_queues -dr TEST_DIR
&quit
```

TEST MODE COMMANDS

The following is a list of the test mode commands. They can be entered from the console only. Complete command descriptions are available in Appendix D of the *Bulk I/O* manual.

coord

command to the driver allows the coordinator part of the test process to come to command level. Reactivate the driver via the start command.

debug

calls the system debug command for setting/resetting breakspoints, executing interactive Multics commands, etc. Available from coordinator or driver command levels.

driver

command to the coordinator creates the driver part of the user's test process.

pi

command to the driver generates a program_interrupt signal. Useful for returning to the debug command after one of its functions has been interrupted by a quit signal.

resume

command to the driver directs it to attempt recovery from iodd signal command level, or return to normal command level from request or quit command level (aborting any current request), as if it were not in test mode.

return

command to the driver logs out the driver without notifying the coordinator or displaying any messages. This command to the coordinator is the same as the logout command.

SECTION 8

CHECKLIST

The following list is a brief repetition of the basic steps for HASP setup. It also includes some reminders about important details. The major items do not have to be done exactly in the order presented.

- Bind bsc_tables and hasp_tables modules into core image before loading FNP:

  1. Produce object segments for bsc_tables.map355 and hasp_tables.map355 via map355 command (if not already in object archive).

  2. Extract FNP's other object segments from archive into directory in search list.

  3. Modify bindfile and produce core image to be loaded via bind_fnp command.

  4. Store core image segment where core image pathname in CDT specifies, by changing pathname in CMF or copying image to place already specified in CMF.

- Decide workstation configuration:

  1. Configure one operator's console.

  2. Configure a maximum of eight card readers.

  3. Configure a maximum combined total of eight line printers and card punches.

- Edit Channel Master File:

  1. Define HASP multiplexer channel with:

     ```
     name: <CHANNEL_NAME>;
       service: multiplexer;
       multiplexer_type: hasp;
       line_type: BSC;
       terminal_type: <HASP_WS_NAME or HASP_H_NAME>;
     ```

  2. Define operator's console subchannel with:

     ```
     name: <CHANNEL_NAME.opr>;
       service: slave;
     ```

  3. Define 0-8 card reader subchannels with:

     ```
     name: <CHANNEL_NAME.rdr1{-CHANNEL_NAME.rdrN}>;
       service: slave;
     ```

  4. Define 0-8 line printers (combined total of printers and card punches cannot exceed 8 and no sum of printer/punch addresses can equal 9) with:

     ```
     name: <CHANNEL_NAME.prt1{-CHANNEL_NAME.prtN}>;
       service: slave;
     ```

  5. Define 0-8 card punches (combined total of punches and line printers cannot exceed 8) with:

     ```
     name: <CHANNEL_NAME.pun1{-CHANNEL_NAME.punN}>;
       service: slave;
     ```

  6. Convert CMF to new CDT via cv_cmf command.

  7. Install CDT via install command.

- Edit Terminal Type File:

  1. Supply terminal type name as given for HASP multiplexer channel terminal_type:

     ```
     terminal_type: <HASP_WS_NAME or HASP_H_NAME>;
      additional_info: "<SEE_NEXT_ITEM>";
     ```

2. Supply additional_info parameters and values if you desire other than the default values. Parameters of most interest to HASP users and their defaults are:

```
type=workstation, block_size=400, signon_mode=no,
multipleave_mode=yes, suspend_all_mode=no,
rts_mode=no, connect_timeout=30,
receive_timeout=3, transmit_timeout=2, max_naks=10,
max_device_input_records=6
```

3. Define a new terminal type with suitable input and output translation tables if you plan to specify the new one in the hasp_host_ or hasp_workstation_ I/O module attach description.

4. Convert TTF to new TTT via cv_ttf command.

5. Install TTT via install command

- Create Access Control Segments for each subchannel. Set access to rw for the attaching processes.

- Edit Project Master File:

1. Register the processes that will attach each subchannel in the PMF; process attaching a simulated operator's console should not be registered on the SysDaemon project.

2. Give processes that will attach each subchannel the dialok and ^vinitproc attributes. Give iod_overseer_ as the initproc pathname.

3. Convert PMF to new PDT via cv_pmf command.

4 Install PDT via install command.

- Edit I/O daemon table:

1.a. When Multics is host, define a major device for each workstation device with:

```
line: <SPECIFY EXACTLY DEVICE'S SUBCHANNEL>;
driver_module_:  remote_driver_;
args: "station= <STATION_ID>, slave=no,
       desc= -term hasp_workstation_ -comm hasp";
```

1.b. When is Multics is simulating a workstation, define a major device for each simulated device, except the operator's console, with:

```
line: <SPECIFY EXACTLY DEVICE'S SUBCHANNEL>;
driver_module_:  hasp_ws_sim_device_;
args: "station= <STATION_ID>, desc= -term hasp_host_
      -comm hasp";
```

Include -signon in args substatement if SIGNON record is required.

Define one minor device for each major device with:

```
minor_args: "dev=<reader_out/printer_in/punch_in>";
```

Additional keywords are optional; however, if you use "request_type=" in minor_args substatement, you must define the Request_type specified with:

```
generic_type: <printer/punch>;
```

2. Define a request type info segment for each printer request type specifies with the rqti_seg substatement; this is optional otherwise. Place in >ddd>iod>rqt_info_segs directory with sma directory access for the administrator, s access for other users. The segments should have rw access for the administrator, r access for the users. Create printer rqti segments via cv_prt_rqti command.

3. Compile I/O daemon table via iod_tables_compiler command.

4. Create the I/O daemon queues via the create_daemon_queues command.

5. Set up message coordinator routings. Create the input and output message segments, then establish virtual consoles and message coordinator sources via the define and route commands (usually in system_start_up.ec).

- Load FNP, if necessary, via load_mpx command, and follow operating instructions described in Section 4 to begin processing.

INDEX

S

T

W

X

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE MULTICS HASP SERVICE AND UTILITY MANUAL

ORDER NO. GB60-00

DATED OCTOBER 1983

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

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE