HONEYWELL

MULTICS COMMUNICATIONS ADMINISTRATION

SOFTWARE

MULTICS

COMMUNICATIONS ADMINISTRATION

SUBJECT

Information Needed by System Administrators for the Management of the Multics Communication System and Communications Channels

SPECIAL INSTRUCTIONS

This publication supersedes the Multics Administrator's Manual – Communications, Order No.: CC75-01, dated July 1982 and its associated addenda, CC75-01A, dated February 1983, and CC75-01B, dated December 1983. Effective with this edition the document is retitled Multics Communications Administration.

Change bars indicate new and changed information; asterisks denote deletions.

See the "Significant Changes" section in the Preface for a description of changed information.

SOFTWARE SUPPORTED

Multics Software Release 11.0

ORDER NUMBER

CC75-02

February 1985

Honeywell

PREFACE

This manual is a guide to the administration of the Multics Communication System. This manual includes information on terminal types, line types, and channel management.

The *Multics Programmer's Reference Manual*, Order No.: AG91, contains general user information about terminal use and communications input/output. Readers of this manual should be familiar with the information in the Programmer's Reference Manual.

Significant Changes in CC75-02

Support for the following hardware features has been discontinued. References have been removed from the documentation.

```
DATANET 6632
DATANET 355
Advanced Remote Display Station (ARDS)
IBM 1050 Model Terminal
IBM 2741 Model Terminal
Bell 202 ETX Modem Protocol
```

The FNP must now be configured by means of a prph configuration card (not an FNP card).

There is additional information concerning hardware flow control using the CTS Dataset lead.

File No.: 1L63, 1U63 CC75-02

The authorization of users allowed to use a communications channel can now be expressed by a range of values (instead of a maximum value only).

A new CMF statement - check_acs - has been added.

A new X.25 TTF parameter - breakall_idle_timer - has been added.

CONTENTS

SECTION 1

OVERVIEW OF MULTICS COMMUNICATION SYSTEM

The Multics Communication System effects the transfer of data between the Multics virtual memory and various remote devices (primarily terminals) over communications channels. This manual is concerned with the Multics Communication System as it appears to a system administrator, and it also discusses the specification and management of channels. *

The bulk of the Multics Communication System resides in the Multics supervisor and in a separate machine, the Front-End Network Processor (FNP). There may be up to eight FNPs on a Multics system. The user-ring and supervisor portions of the Multics Communication System are principally concerned with terminal management, while the primary responsibility of the FNP is channel management. The determination of the number and types of channels to manage, however, comes in part from the physical configuration, and in part from a user-ring data base, the channel definition table (CDT). The CDT is maintained by the system administrator; its contents are described in Section 4.

TERMINALS AND CHANNELS

The term "channel" (or "communications channel"), as used in this manual, refers to the logical connection between the system and a remote input/output device via an FNP. This includes a physical connection, which may go through a telephone system or a private communications network, or may consist of one or more hardwired cables. Normally, there is a one-to-one correspondence between logical connections and physical connections, except in the case of multiplexed channels, described later in this section.

The word "terminal" is used to refer to the device itself; it may be an ordinary interactive terminal, or it may be a computer controlling one or more peripheral devices. This manual is concerned primarily with channels and channel management; the use and control of terminals is described in the *Multics Programmer's Reference Manual*, Order No.: AG91.

FNPS, CONTROLLERS, ADAPTERS, AND SUBCHANNELS

The Multics Communication System can be configured with any model of the DATANET 6670 series of Front-End Network Processors that includes at least 64K words of memory (DATANET 6661 and DATANET 6678).

An FNP is connected to a Multics input/output multiplexer (IOM) by means of a peripheral of the FNP called the direct interface adapter (DIA), and a DI channel in the IOM. A Datanet 6670 interfaces to communications channels through a device known as a high-speed multiline controller, or HMLC. The controller has subchannels to which the individual communications channels (sometimes called "lines") are connected.

An HMLC handles synchronous channels at speeds ranging from 1,200 to 72,000 baud, or asynchronous channels at speeds ranging from 110 to 19,200 baud. A Datanet 6670 can have up to 12 HMLCs, with each one capable of supporting up to eight subchannels (with a maximum combined speed of 72,000 baud). For compatibility, HMLCs are discussed in this manual in terms of HSLAs (high speed line adapters). Thus, four HMLCs are the equivalent of one HSLA, since an HSLA can support up to 32 subchannels.

MULTIPLEXED CHANNELS

The device associated with a physical channel may be some kind of concentrator that controls multiple terminals (examples of such concentrators include the Honeywell VIP 7700 series and the IBM Model 3270 series). For some types of concentrator, the Multics Communication System is capable of treating each terminal as a separate logical channel, if so instructed by the CDT. In this case, the channel occupied by the concentrator is called a multiplexed channel, and the concentrator is referred to as a multiplexer; the logical channels associated with the individual terminals are called subchannels of the multiplexer. The multiplexed channel and its subchannels must all be defined in the CDT. In theory, a subchannel of a multiplexer might itself be multiplexed. Such multiplexing can be carried to any number of levels.

Since an FNP controls many channels, but communicates with the central system over a single channel (the DIA), the FNP may be regarded as a multiplexer. In fact, as indicated in Section 5, the operator commands used to control multiplexers (load_mpx, start_mpx, etc.) are also used to control FNPs. Furthermore, frequent reference is made in this manual to a channel's multiplexer or "parent multiplexer" (i.e., the multiplexer of which it is a subchannel); when the channel referred to is a physical FNP channel, the parent multiplexer is the FNP.

Appendix A contains detailed information on how to configure certain system-supplied multiplexers.

CHANNEL NAMES

Each communications channel has a unique name that identifies the FNP, controller or adapter, and subchannel through which it is connected. Subchannels of multiplexed channels are identified by additional components in their names. For example, "a.h102" identifies channel 02 of HSLA 1 on FNP a. The format of a channel name is described in detail in the *Multics Programmer's Reference Manual*, (Order No.: AG91).

INITIALIZATION

When the system comes up and the answering service is initialized, each FNP specified in the CDT is loaded (provided that a corresponding prph card appears in the config deck, and that the CDT does not specify a service type of "inactive" for the FNP). If the FNP is successfully loaded, all the communications channels on that FNP are initialized. For nonmultiplexed channels, this means "listening" to the channel, i.e., putting it in a state in which dialups are possible. For multiplexed channels, initialization means setting up various internal data bases and initializing each subchannel of the multiplexer.

If a multiplexer hangs up, all its subchannels are automatically hung up, and the multiplexer (and its subchannels) is reinitialized. Similarly, if an FNP crashes, it is automatically reloaded, and all its channels reinitialized. For information on how to control the loading and initialization of multiplexers (including FNPs), see Sections 5 and 6.

CONSISTENCY OF CONFIGURATION

In order for a channel or set of channels to be usable, the data bases describing the channel configuration must be consistent with each other, and with the physical configuration. In particular, the following points should be observed:

- All channels must be represented in the channel definition table (CDT). If channels are missing from the CDT, they can be added and the CDT replaced as described in Sections 4 and 5; use of such added channels requires the reloading of a multiplexer, as described in Section 5.

- The attributes and configuration of each channel as described in the CDT must agree with its actual characteristics; specifically, it must be cabled to the correct subchannel of the correct controller, and the appropriate modems (if any) must be installed with the correct options (see Section 3 for more information about modems). If the CDT and the configuration disagree, either the CDT should be corrected (as indicated above) or the physical configuration should be adjusted to agree with the CDT. In the latter case, the multiplexer probably does not need to be reloaded; however, the channel may have to be reinitialized by using the attach and detach commands (see Section 5).

- Any FNP that is to be used must be represented by a prph card in the config deck, and must be cabled to the IOM channel specified on the prph card. (The config deck is described in the *Multics System Maintenance Procedures* manual, Order No.: AM81.) If the prph card for the FNP is specified incorrectly or is missing, the FNP is unusable until the next Multics bootload; the config deck has to be corrected between bootloads. See the *Multics System Maintenance Procedures* manual, Order No.: AM81 for more information.

- Each FNP must also be represented by an entry in the CDT; the FNP core image specified in this entry must exist, and must support at least as many controllers as are specified in the CDT entry. If any channels on that FNP require special communications protocols (see Section 3), the FNP modules that implement those protocols must be included in the core image (see Section 6). If the core image and the CDT entry disagree, either the CDT should be replaced, or the core image should be recreated using the bind_fnp command (described in Section 7). In either case, the FNP must be reloaded before use.

- A site may implement its own communications protocol(s) and add its own FNP control tables module(s) to the core image. The line types ASYNC_1, ASYNC_2, ASYNC_3, SYNC_1, SYNC_2, and SYNC_3 are reserved for such site-defined protocols.

- All terminal types specified in the CDT, as well as any others that the site intends to support, must be represented by entries in the terminal type table (TTT). A standard TTT is supplied with the system; however, if the site has terminals other than those specified in the standard TTT, the system administrator may add entries for those terminals to the terminal type file (TTF) and replace the system TTT (>system_control_1>ttt). In certain cases, individual users may wish to define their own terminal types in a private TTT; for this reason, the TTF and TTT are described in the *Multics Programmer's Reference Manual*, (Order No.: AG91).

SECTION 2

TERMINAL TYPES AND LINE TYPES

CLASSIFICATION OF COMMUNICATIONS CHANNELS

The Multics Communication System is designed to support a wide variety of communications protocols and devices. For this reason, it is necessary to identify the particular characteristics of each channel. This identification is accomplished by assigning a line type and a terminal type to each channel.

A line type defines the communications protocol used by a channel. The following items are aspects of the line type:

- synchronous or asynchronous transmission
- character size
- error detection and recovery
- message blocking
- message acknowledgement
- modem control

Not all line types have all of the above aspects. Only synchronous line types, for example, are concerned with messages, which are blocks of continuously transmitted characters (see the discussion of synchronous protocols in Section 3).

A terminal type defines the operating characteristics of a terminal. These characteristics include:

- character set (graphic symbols represented)
- character code (e.g., ASCII, EBCDIC, etc.)
- control sequences required to effect carriage movement and other special functions
- number of fill characters required for carriage movement functions
- terminal initialization sequence to clear screen, set tabs, etc.
- line length and page length
- character echoing modes

The terminal type concept permits the Multics Communication System to provide a uniform interface for terminal input/output. This eliminates the need to embed terminal-specific knowledge in most user and system software.

LINE TYPES

The following standard line types are provided by the Multics Communication System:

*	ASCII	device similar to 7-bit ASCII using Bell 153-type modem protocol
	SYNC	ASCII synchronous connections, no protocol
	G115	ASCII synchronous connection, Model Level 6 remote computer interface (RCI) protocol
*	BSC	binary synchronous protocol
	VIP	device similar to Honeywell Model 7700 Visual Information Projection (VIP) system
	POLLED_VIP	device similar to Honeywell Model 7760 VIP system
	X25LAP	synchronous High-level Data Link Control (HDLC) X.25 protocol implementing frame level of CCITT recommendation of 1980. Used mainly to interface with a Value Added Network (VAN) such as TYMNET.

It is possible for a site to create new line types not already provided by the Multics Communication System. Each such new line type requires the addition of a site-provided control tables module to the FNP software. This module must implement the desired
* communications protocol.

Line Type Assignment

The assignment of line types to channels is specified in the channel definition table (CDT). A complete description of the CDT is contained in Section 4 of this manual. The selection of a line type for a given channel must, of course, be consistent with the actual channel hardware. If no line type is specified for a channel, one of the several asynchronous line types is automatically selected. Therefore, line types should always be specified for synchronous channels.

The line type of a channel can only be changed while the channel is not in use. Ordinary users cannot do this. In order to change a line type, an administrator must create a new CDT specifying a new line type for a particular channel. The new CDT must then be installed. The line has to be removed from the system using the operator command, detach, and reattached using the attach command. If the channel is not dialed up at this time, the new line type takes effect immediately. Otherwise, the new line type does not take effect until the current connection is terminated.

TERMINAL TYPES

The terminal types provided by the Multics Communication System are defined in a segment called the terminal type table (TTT). This segment is generated from a source segment called the terminal type file (TTF). A complete description of the TTF and the TTT is contained in the *Multics Programmer's Reference Manual*, Order No.: AG91.

A standard version of the TTF is provided with each system release. It has the pathname >tools>TTF and is copied to >udd>sa>a at site initialization time. This TTF can be easily modified to add new terminal types or to change characteristics of existing terminal types. This allows each site to develop a customized TTF and TTT.

Although most users ordinarily depend on the site-installed version of the TTT, it is possible for any user to substitute his own private TTT. A user can create a TTT in exactly the same manner as a system administrator. This affords users the ability to define terminal types suited to their individual preferences.

Terminal Type Assignment

An initial terminal type is assigned to a login service channel at dialup time in one of several ways. An initial terminal type can be specified in the CDT. This is commonly done for dedicated channels which are always connected to the same terminal. If an initial terminal type is not specified in the CDT, then the default type table in the TTT is used to choose a terminal type based on line type and baud rate. In either case, an attempt is made to read an answerback (if allowed by the CDT) from the terminal. If the terminal responds, the answerback is decoded according to answerback specifications in the TTT which indicate initial terminal types.

A terminal type for slave or autocall channels may be specified in the CDT. If this is done, then the terminal type and the default modes for the terminal are set and the initial string is sent at slave channel dialup time or when an autocall channel is connected. If a terminal type for slave or autocall channels is not specified in the CDT, then the above steps are not performed.

The initial terminal type established by the system can be changed by the user. The terminal_type (ttp) preaccess command can be used prior to login to alter the established terminal type, or the user can specify the -terminal_type control argument to the login command to set the terminal type accordingly. After login, the user may invoke the set_tty command to change his terminal type. The user may also change his TTT by use of the set_ttt_path command. Subsequent use of the set_tty command will then reference the user-selected TTT. All of these commands are described in the *Multics Commands and Active Functions* manual, Order No.: AG92.

SECTION 3

MODEMS AND TERMINAL CONNECTIONS

A terminal or concentrator can be connected to an HMLC in any of several ways, * depending on the line type and the available equipment. The path between the FNP and the terminal is often referred to as a "communications link." This section describes the general types of communications links that are possible, and includes a list of communications equipment that may be used to connect a device to a Multics system.

TYPES OF COMMUNICATIONS LINKS

Communications links can be categorized in a variety of ways. The following paragraphs discuss the most important distinctions among communications links, namely, whether they are: asynchronous or synchronous; full duplex or half duplex; hardwired, private-line, or dialup.

Asynchronous/Synchronous

An asynchronous link is one over which characters are transmitted singly, at unpredictable intervals. The individual bits of each character are transmitted at a fixed rate (the bit rate or baud rate of the channel), but the time between characters is arbitrary. To allow the hardware to identify character boundaries, each character is preceded by a start bit and followed by one or two stop bits. Each character normally includes a parity bit, which is used to check whether the character has been received correctly.

On a synchronous link, characters are transmitted in continuous blocks, and the time between blocks is arbitrary. One or more instances of a special bit pattern, called a synchronization character, is transmitted at the beginning of each block. Parity bits may or may not be included; some kind of checksum is generally appended to each block to ensure correct reception of the block. The blocks transmitted over a given link often must contain control information in some specified format; see the discussion under "Communications Protocols" below.

Interactive terminals generally use asynchronous transmission; remote computers, printers, concentrators, etc., generally require a synchronous link.

Full Duplex/Half Duplex

A communications link is full duplex if data can be transmitted across it in either direction, or both directions, without changing the physical state of the link. It is half duplex if the connection permits transmission in only one direction at a time, and requires specific action to alter the direction of transmission (known as "turning the line around"). Because of the number of telephone wires required to implement the link, full duplex links are sometimes called "4-wire", and half duplex links are called "2-wire." Asynchronous links are normally full duplex; synchronous links may be either full duplex or half duplex.

Hardwired, Private Line, Dialup

A hardwired link is one in which the terminal is connected directly to the FNP by means of an appropriately wired cable, i.e., a DC path. The length of the cable is normally limited to 50 feet, but can be longer, depending on speed, cable type, and hardware at each end.

A private line is a link that utilizes a telephone-like transmission medium, and requires a modem at each end (see the discussion of modems below); however, the physical connection is reserved for use by the specific terminal-FNP link, and does not have to be reestablished for each use. The path may be furnished by the telephone company (an AC path), or be a long wire (a DC path).

A dialup link uses a switched network, usually the public telephone system. Each such link must be established by dialing into the network at the start of a terminal session, and is closed by a hangup operation. Thus, while a hardwired or private-line channel appears to be dialed up immediately whenever a listen operation is performed by the system, a dialup channel is not available for use until a user has actually dialed the phone.

Hardwired and private-line links are usually full duplex; dialup synchronous links may be either half duplex or full duplex.

It should be noted that when a channel with a login service type dials up, if no commands are entered within a reasonable time interval (normally two minutes) the answering service automatically hangs up the channel and listens to it again. This is not desirable on a hardwired or private-line channel, since another dialup signal follows immediately; therefore, the hardwired attribute in the CDT entry for the channel instructs the answering service to leave the channel dialed up until the channel starts being used. This is the only effect of the hardwired attribute in the CDT. (See the description of the channel master file in Section 4 for more information about service types and channel attributes.)

The private_line keyword used in the dataset statement in the CMF indicates that the channel is a private-line link, and also that it is full duplex. The private_line keyword is only interpreted if the dataset specified is "201C". Other datasets default to either private-line or dialup; see Table 3-1 (later in this section) for more information.

MODEMS

A modem (modulator/demodulator) is a device that converts a digital signal (received and generated by computers and terminal equipment) to an analog signal (suitable for transmission across telephone lines) and vice versa. A communications link that is not hardwired requires two modems, one cabled to the FNP and one cabled to the terminal, with a telephone-system connection between the two modems. Dataset is a frequently used synonym for modem.

There are many different types of modems: synchronous and asynchronous; with and without dialing capabilities. Some can run at a variety of speeds, and some are restricted to a specific speed. The modems currently most common are those manufactured by American Telephone & Telegraph. Modems made by other manufacturers are available, although they generally are approximate equivalents of Bell modems. A specific modem model may be obtained with a variety of options. In most cases, the options recommended by the manufacturer should be used, but a few exceptions are noted in Table 3-1.

The two modems at the ends of a communications link must be either the same type or compatible types and have similar or compatible options. In a few cases, pairs of modems operate by having one member initiate the call (at the terminal end) and the other respond to it (at the FNP end); the Bell Model 113A/113B is an example of such a pair.

Table 3-1 lists Bell modems currently usable on a Multics system, and indicates, for each one, the speed(s) at which it can run, whether or not it is synchronous or asynchronous, whether it is dialup or private-line or both, what specification should be given in the dataset statement of the CMF, and any other important information. For more detailed information on a particular modem, refer to the modem's technical specification, which is available from the manufacturer.

COMMUNICATIONS PROTOCOLS

A communications protocol is a set of conventions observed by the two ends of a communications link in order to exchange data in an orderly and predictable fashion. The specification of a protocol may include conventions for controlling the physical state of the link, as well as the format and sequencing of blocks of data. Full duplex asynchronous links do not generally use protocols; half duplex asynchronous links usually require fairly simple protocols to determine the direction of transmission at any given time. Synchronous links generally use more elaborate protocols to ensure that each block of data is received correctly and in correct sequence.

* The following are examples of protocols supported by the Multics Communications System, with the line types used to implement them (see Section 2 for a further discussion of line types).

*

Examples of Protocols

Protocol: Level 6 RCI
Line Type: G115

Protocol: Binary synchronous (bisync)
Line Type: BSC

Protocol: Honeywell polled VIP
Line Type: POLLED_VIP

Table 3-1. Modem Descriptions

Model No.	Speeds	Sync/ Async	Duplex	Private/ Dialup	CMF specification	Remarks
103A	up to 300 bps	async	full	dialup	103A or none	
113A	up to 300 bps	async	full	dialup	103A or none	originates calls only
113B	up to 300 bps	async	full	dialup	103A or none	answers calls only
212	up to 1200 bps	async sync	full	dialup	none required	high-speed option should be used
201C	2400 bps	sync	half	dialup	201C	
201A 201B 201C	2000 bps	sync	full	private	201C private-line	
208A	4800 bps	sync	full	private	208A	
208B	4800 bps	sync	half	dialup	208B	
209A	9600 bps	sync	full	private	209A	

AUTOMATIC BAUD RATE DETECTION

The autobaud feature of the Multics Communication System is designed to recognize/configure the baud rate (bits per second) of an asynchronous HMLC (or HSLA) channel at dialup time. The autobaud facility selects a 1200 baud rate if the high-speed * indicator is on (pin 12, for most modem types); otherwise it selects a rate of 110, 150, 300, or 1200 baud based on the sampling of bit changes for the incoming characters, "1", "L", "CR".

Lead Control Selection of 1200 Baud

If the modem connected to the FNP turns "on" pin 12 of the cable connected from it to the FNP, the channel is set to 1200 baud and it is then handled in the normal manner. The FNP modem can be set to respond to a switch on the terminal modem which indicates that the terminal is operating at 1200 baud. The operation of the channel does not appear any different to the user than if a strictly 1200 baud channel is dialed into (there is no requirement for the user to type any characters before the login banner is received).

Bit Sampling Selects Other Bauds

If the signal on pin 12 is "off", sampling for the bit changes of an input character is performed. To accomplish this, the following sequence occurs:

1. The user establishes a connection with the host.

2. The user types in either the letter "1" or "L" or a carriage return.

3. The software in the FNP scans the incoming bit stream looking for bit changes at 300 baud. Since the sample character is known ("1", "L", "CR"), the changes in state of the bits ("0" or "1") indicate the timing necessary to transmit the bits, and therefore the baud rate of the channel is determined.

4. The channel is then handled in the normal manner. The answerback is checked (if required), the initial string (if any) is sent and the login banner is displayed.

5. The user types in any of the preaccess commands (MAP, etc.) if desired.

6. The user logs into the system using "1" again for "login", enter, etc.

Modems

Any asynchronous HMLC channel may be configured with the autobaud feature. For dialup channels not using special modems, automatic baud rate detection is possible only up to 1200 baud. In this range, most modems and acoustic couplers are able to interface with host modems.

There are several modems that can be used on channels configured with the autobaud feature which make use of the pin 12 lead change. Vadic (Model 3467) and Western Electric (Model 212A) are two manufacturers that make such modems. Special encoding techniques are required to handle the 1200-baud full duplex data over voice-grade dialup lines. For this reason, a modem of one manufacturer may not necessarily be able to interface to a modem of another manufacturer.

Hardware Flow Control Using the CTS Dataset Lead

CTS flow control protocol utilizes the capabilities of the FNP's asynchronous communications adaptor to remove the need for delay calculation to manage output flow control and also eliminates the need for output flow control information embedded in the data stream. This flow control is implemented for hardwired asynchronous communications lines (lines utilizing the FNP module 'control_tables'). It provides a stop-on-character output flow control.

This protocol utilizes the CTS dataset lead (pin 5) to control output from the FNP to the Data Termination Equipment (DTE). If CTS is high, output will be sent; when CTS drops, the current character will be finished and output will cease until CTS is raised again.

Ths protocol is implemented such that a line must have all three leads (CTS, CD and DSR) high to be initially on-line. After this point CTS will act as a flow control signal, until the line is hungup again by dropping either CD or DSR.

Many terminals can utilize this protocol or a DTR flow control protocol. The DTR protocol uses the DTR lead from the terminal (pin 20) in the same manner. For DTR flow control a connector must be wired which connects the terminal DTR to the FNP CTS.

The use of a hardware protocol removes computational loading from the mainframe MCS since delays do not need to be calculated. It also lightens buffer loadings since buffer space for delay characters is not needed.

SECTION 4

CHANNEL MASTER FILE

CHANNEL DEFINITION TABLE

The channel definition table (CDT), a data base in the segment:

```
>sc1>cdt
```

describes the attributes of all Datanet Front-End Network Processors (FNPs) and communications channels on the system. Write access on this segment is necessary only for the initializer; read access is necessary for users of certain metering commands.

The CDT is a binary table containing numbers and pointers as well as character strings. Therefore, it cannot be examined or modified with standard editors. The display_cdt command is used to print the contents of the whole CDT or selected entries. The reset_cdt_meters command resets the per channel count of dialups and total connect time. The tty_lines command lists all channels and their counters. When the system administrator wishes to add or delete channels, or to change the information about a channel in its CDT entry, he modifies the channel master file (CMF), converts the CMF into a new copy of the CDT with the cv_cmf command, and uses the install command to signal the answering service to modify the system's copy of the CDT. Thus, the site management can make certain changes in the site's configuration of terminal channels without waiting for a system shutdown.

Each FNP and terminal channel in the system has a CDT entry. This includes not only channels used for logins, but also special terminal channels, such as channels used by the message coordinator for the I/O of the initializer and daemons, and channels describing remote stations operated by the I/O daemons. These channels have flags set in the CDT entry indicating the way they can be used.

CHANNEL MASTER FILE

The CMF describes all FNPs, multiplexers, and terminal channels on the system. It is an ASCII file, normally stored in >udd>sa>a>CMF.cmf, that can be converted into a binary table, which is installed by the answering service at the system administrator's request.

Syntax of the CMF

The CMF consists of a series of entries, one for each FNP or channel. Each entry consists of a series of statements that begin with a keyword and end with a semicolon. White space and PL/I-style comments enclosed by /* and */ may appear between any elements of the CMF. The last entry in the CMF must be the end statement. Global statements specifying defaults appear anywhere before the end statement; the defaults that they specify are in effect for all subsequent channel entries, until they are changed by subsequent global statements. Except for the end statement, all statements consist of the statement keyword, a colon, the variable field of the statement, and a semicolon.

The entry for each FNP consists of an FNP statement, followed by statements describing the attributes of the FNP. Attributes not specified for an FNP are set from defaults supplied by the cv_cmf program.

The entry for each channel consists of a name statement naming the channel, which may be followed by statements describing the attributes of the channel. Attributes not specified for a channel are set from the defaults established by global statements or those supplied by the cv_cmf command. (See "CMF Default Statements" below.) Thus, a very simple CMF could consist of an FNP entry, a name statement for each channel, and an end statement, as follows:

```
FNP:     A;
type:    DN6670;
memory:  64;
hsla:    1;
name:    a.h000;
name:    a.h001;
end;
```

FNP ENTRIES IN THE CMF

A description of each statement in an FNP entry of the CMF is given below.

FNP: <name>;
The FNP statement is required. It specifies the tag of the FNP being described. Up to eight FNPs may be specified. A prph card with the same name is also required in the config deck. See the *Multics Programmer's Reference Manual*, Order No.: AG91, for restrictions on the name of an FNP.

type: <FNP type>;
The type statement is required. It specifies the type of the FNP, which must be DN6670.

memory: <K-words>;
The memory statement is optional. It specifies the size of the FNP memory in K (1024 words). Acceptable values are 64 through 256, in increments of 32 (i.e., 64, 96, 128, etc.) The default is 64.

hsla: <number of hslas>;
The hsla statement is optional. It specifies the number of HSLAs (the number of HMLCs divided by four) configured on this FNP. The number may be zero through three; the default is zero.

image: <path>;
The image statement is optional. It specifies the pathname of the FNP core image to be used for this FNP. The default is >system_control_1>mcs. Each FNP may have a different core image depending on its configuration and supported line types. The path argument should specify a segment that is on the root logical volume to ensure that the FNP can be reloaded without having to worry about demounted volumes. (The FNP cannot be reloaded if path is on a demounted volume.)

service: <active or inactive>;
The service statement is optional. If specified, the variable field must be either active or inactive. An active FNP is loaded and used by the system. An inactive FNP is not automatically loaded. However, it can be loaded by operator command or made active by installation of a new CDT. (If an FNP is missing from the CMF at answering service startup time, rather than being included with a service type of inactive, it cannot be used during the current Multics bootload. The default is active. See "Changing Channel Configuration," below.)

CHANNEL ENTRIES IN THE CMF

A description of each statement in a channel entry of the CMF is given below.

name: <channel name>;
The name statement is required. It specifies a unique channel name for the channel and can be up to 32 characters long. FNP-type channel names are divided into components separated by periods, with each component representing a level of multiplexing. A complete description of channel naming conventions appears in the *Multics Programmer's Reference Manual*, Order No.: AG91.

name: <channel name 1>-<channel name 2>;
A pair of channel names can be given in the name statement. The first N characters of the two names must be identical, where N is that portion of the names up to the digits representing the first and last subchannel numbers (the subchannel number in the second name must be greater than that in the first name). The effect of this form of the name statement is to specify a group of adjacent channels having identical characteristics. Their names are formed by starting with the first name, increasing the subchannel number by 1 to form each name, and ending with the second name. The number of channels in the group is equal to one plus the difference between the two subchannel numbers.

For example: the name statement a.h100-a.h220 specifies a group of successive channel names, the first of which is a.h100, the second, a.h101, the third, a.h102, etc., up to a.h220. Likewise, the name statement b.h203.prt1-b.h203prt8 specifies a group of eight channel names, the first of which is b.h203.prt1, the second, b.h203prt2, the third, b.h203.prt3, etc., up to b.h203.prt8.

generic_destination: <"string">;

The generic_destination statement is optional and applies only to the slave and autocall service types. The string may be any site-specific value (e.g., tymnet, modem, protocol_converter_box) that does not resemble a channel name. If included in the CMF entry, the string is used to match the channel specifier used in a dial_out or privileged_attach operation.

baud: <baud_rate>;

The baud statement is optional. It specifies the baud rate of the channel. The baud rate must be compatible with the FNP line adapter type and must be chosen from the following list:

0	600	7200
110	1200	9600
133	1800	19200
150	2400	40800
300	4800	50000
		72000

The channel is assumed synchronous only if a synchronous line type is also specified. Zero and none are equivalent and may only be used for network channels, which are not attached through an FNP. The auto keyword requests automatic baud rate detection at dialup time for HSLA channels (see the discussion of automatic baud rate detection in Section 3). Baud rates higher than 19200 are valid only for synchronous channels.

NOTE: At speeds of 4800 baud and below, no more than 960 characters are sent to the FNP at a time.

The baud rate of a synchronous channel is not controlled by software. The baud statement for a synchronous channel simply informs the system of the speed of the hardware connection. The correct baud rate should be specified, however, so that the system can choose the optimal buffer size for the given rate.

attributes: <attr1, attr2, ... attrN>;

The attributes statement is optional. This statement applies only to login service channels. When this statement is used, the global attributes are overridden for this channel. If no attributes statement is specified, the global attributes are used. The ^ character is used to negate the specified attribute. The following attributes may be specified:

audit, ^audit
: if enabled, access errors on the channel are audited.

* check_answerback, ^check_answerback
: if enabled, specifies that the answerback returned by the channel is to be checked against the one supplied in the answerback statement (see below).

dont_read_answerback, ^dont_read_answerback
: if enabled, specifies that no attempt is to be made to read the answerback of the channel.

hardwired, ^hardwired
if enabled, specifies that the channel is directly wired, not connected through a switching system.

none
if enabled, specifies that all of the "^" values of the attributes listed above are assigned.

set_modes, ^set_modes
if enabled, specifies that the modes associated with the terminal type of the channel are to be set when the channel dials up.

access_class: <"authorization{:authorization}">
The access_class statement is optional. This statement specifies the authorization (sensitivity level and category set) of users allowed to use this channel. The authorization can be specified as a single value, in which case the channel is usable only by users with the specified authorization. The authorization can be specified by a minimum and maximum value, in which case the channel is usable only by users with an authorization equal to or greater than the minimum value and equal to or less than the maximum value. If specified, the authorization must be enclosed in quotes and must be expressed using valid site-defined authorization strings. (Use the print_auth_names command for a list of valid authorization values. See the *Multics Programmer's Reference Manual* Order No.: AG91, for a detailed description of the AIM mechanism.) For example, to specify that a channel is to be used only by users with an authorization of L1,C1, the access class entry would be specified as follows:

```
access_class: "L1,C1"
```

To specify that a channel is to be used by users with an authorization in the range system_low to L5,C2, the access class entry would be specified as follows:

```
access_class:  "system_low:L5,C2"
```

If the access class statement is not specified, the value is assumed to be that specified (or defaulted to) by the Access_class global statement.

The administrator must be aware that the system cannot establish the authorization of a user for any channel except channels with a multiplexer_type value of "sty." For this reason, it is recommended that all channels except those identified as multiplexer_type sty should be specified with a single access_class value. The only exception to this recommendation is for channels with a service_type value of "login" . These channels should be assigned an authorization range sufficient to cover the users who are to be permitted to log in over the channel.

answerback: <string>;
The answerback statement is optional. This statement applies only to login service channels. If specified, it must have as a variable field a four-character string that is the expected answerback for the channel. The star convention is allowed. If the value is null, then no answerback check is made. Otherwise, the answerback code from the terminal is matched with the specified field; if they disagree, a message is printed to the user and to the operator, and the channel is hung up.

terminal_type: <terminal type>;

The terminal_type statement is optional. If specified, its variable field must contain one of the terminal types defined in the terminal type table (described in the *Multics Programmer's Reference Manual*, Order No.: AG91); specified terminal types are checked against the current TTT when the CDT is installed.

The names of the possible terminal types can be ascertained by using the print_terminal_types command (described in the Commands manual). These names may be entered in uppercase or lowercase. The terminal type specified is set before any input or output is done to the terminal; thus, this statement overrides the default
* terminal type (which is based on the line type and baud rate). The terminal_type statement does not force the terminal to remain the same type; it merely specifies the initial type. Thus, the answerback identifier, the -terminal_type login control argument, and process requests may override the terminal_type statement.

line_type: <line type>, <attr1>, <attr2>;

The line_type statement is optional. If specified, its line type field must contain one of the following line types:

```
none          unspecified (select protocol at dialup time)
ASCII         terminals:  ASCII, TN300, TTY33, TTY37, TTY38
*  Sync       synchronous line, no protocol
G115          remote computer interface, synchronous (GRTS)
BSC           binary synchronous communications
*  VIP        Honeywell VIP 7700, single station
POLLED_VIP    Honeywell VIP 7760
X25LAP        HDLC X.25 frame level

ASYNC1
ASYNC2        optional site-defined asynchronous line types
ASYNC3

SYNC1
SYNC2         optional site-defined synchronous line types
SYNC3
```

The line type specified is passed to the terminal control software before the channel is enabled for listening, in order to set the line protocol to be used for handling
| communications on the channel. The default is ASCII.

*

dataset: <name>, <attr>;

The dataset statement is optional. It identifies the generic type of dataset/modem being used on the channel. If specified, the name field must be chosen from the following list of Bell modem names, although any modem that is similar may be used.

```
103A          208A
201C          208B
202C5         209A
202C6
```

The attribute may be private_line to indicate that the modem is being used on a 4-wire circuit (see Section 3 for more information).

charge: <chargename or none>;

The charge statement is optional. If specified, its variable field must contain a charge name listed in the device prices array in the installation_parms segment (See the *Multics System Administration Procedures* manual, Order No.: AK50 for more information), or be "none." The charge type selects the per-hour surcharge for use of the channel.

service: <service type>;

The service statement is optional. If specified, its variable field must be one of the following service types:

```
login
mc
slave
autocall
inactive
multiplexer
```

The service type determines whether the channel can be used for normal logins or is under control of some special software. The slave service is used only for special channels attached by the I/O daemons for bulk data input/output or other special subsystems. The mc service is specified for all message coordinator channels. The autocall service indicates that the channel is used to dial out using an automatic call unit. The inactive service indicates that the channel exists but is currently not to be used. A channel installed as inactive can be changed only through installation of a new CDT. A channel rendered inactive because of an error condition or operator command (remove) can be changed either through the operator attach command or installation of a new CDT. The multiplexer service type is used for a multiplexer channel (see the discussion of multiplexers in Section 1). If a service type of multiplexer is specified, the multiplexer_type statement (see below) must be provided.

Access Control Segments (ACSs) specify access to the slave and autocall lines. Access to login lines may be similarly controlled—see the description of the attributes* statement above.

A service type of slave must be specified for every channel that an I/O daemon driver directly attaches, and the names of such channels must be specified in the I/O daemon tables, >ddd>idd>iod_tables. Channels that are to be dialed to a daemon driver must be given a service type of login in the CMF, and dial identifiers must be specified in the I/O daemon tables. (In order to use a particular registered dial identifier, a user must have rw access to the ACS. (See the *Multics System Administration Procedures* manual, Order No.: AK50 for more information.)

The default service type is login—meaning that the normal logins are allowed.

multiplexer_type: <type>, <active or inactive>;

The multiplexer_type statement is required for channels with a service type of multiplexer and is invalid for other channels. If it is specified, the variable field must be ibm3270, vip7760, hasp, sty, or x25. It indicates the type of multiplexer to be connected to the channel. The multiplexer is initialized as soon as its parent multiplexer initializes if this statement includes the active key. The multiplexer is not initialized until the load_mpx operator command is used if this statement includes the inactive key. The default is active.

comment: <"string">;

The comment statement is optional. If its variable field contains any blanks or punctuation, it must be enclosed in quotes. The variable field is saved in the CDT entry (up to 48 characters). The comment field may be used to identify the modem and telephone line connected to a channel. The comment is printed by the tty_lines command and by various answering-service programs that print error messages about the channel.

initial_command: <"initializer request">;

The initial_command statement is optional. This statement applies only to login service channels. If it is specified, the variable field must be a valid initializer command (e.g., login, dial) line, enclosed in quotes. Whenever the channel dials up, the initial command is executed before the first line is read from the terminal.

check_acs: <keyword(s)>

Each keyword (described below) specifies that the channel access control segment (>sc1>rcp>CHANNEL_NAME.acs) is to be checked for appropriate access when the specified operation is performed. One or more keywords can be specified; if multiple keywords are specified, each must be separated by a comma. Use of this keyword assumes the creation of an access contol segment (acs) for the channel. See the *Multics Programmer's Reference Manual*, Order No.: AG91 for additional information on access control segments.

login - The user must have rw access to the acs segment to log in over this channel.

slave - A user entering the "dial" or "slave" preaccess command must specify the -user control argument and must have rw access to the acs segment

priv_attach - The user must have rw access to the acs segment to attach this channel through the dial facility.

dial_in - A user accepting dial-in connections (via dial_manager_$allow_dials_ or dial_manager_$registered_server_) must have rw access on the acs segment of the connecting channel for the channel to be connected to the user's process.

dial_out - The user must have rw access to the acs segment to use the channel for dial_out requests.

all - Use all of the above keywords.

Individual keywords can be preceded by ^ to disable the particular keyword. The example below illustrates use of the disabling feature:

```
Check_acs: all;
      .
      .
      .
name: a.h001;
 check_acs:  ^login
```

This sequence results in acs checking for all cases except logins from channel a.h001.

CMF Default Statements

Control statements that begin with a capital letter are global statements. Global statements exist for each of the statements that specify attributes. If no global statements are given in a CMF, the following set of defaults is assumed:

```
Attributes:           none;
Access_class:         system_low;
Answerback:           "";
Baud:                 300;
Generic_destination:  "";
Line_type:            none;
Terminal_type:        none;
Charge:               none;
Service:              login;
Comment:              "";
Initial_command:      "";
Check_acs             priv_attach, dial_out;
```

GLOBAL STATEMENTS APPLICABLE TO ALL FNPS

The following additional global statements are used to set system-wide parameters:

FNP_required_up_time: <number of minutes>;

The FNP_required_up_time statement is optional. It sets the minimum acceptable time between failures for all FNPs. If an FNP crashes after being up for less than the specified number of minutes, twice in succession, then it is not reloaded automatically. The default is five minutes. A value of zero disables automatic reloading of FNPs.

Spare_channel_count: <number of spare channels>;

The Spare_channel_count statement is optional. It specifies the number of channels the site expects to add to the CMF between system initializations (without rebooting Multics). In other words, if Spare_channel_count is 20, and the CDT in use at system initialization time specifies 100 channels, no more than 120 channels can be configured during that session. The LCT (logical channel table) is allocated in tty_buf with the number of entries equal to the number of channels in the CDT plus the number specified in this statement. An FNP initialization error occurs if the number of new channels being added exceeds the spare channel count and an FNP load occurs before the system is reinitialized. The default is 10.

Changing Channel Configuration

The channel configuration and the characteristics of existing channels can be changed at any time by installing a CDT containing the new information. Section 5 describes the procedures for making these changes and discusses their implications.

The parameters that are changed at the next Multics bootload are actually set during answering service initialization, when either the startup or multics command is given. If necessary, changes to these parameters can be made by copying a new CDT into >sc1 in the initializer process in admin mode, before the answering service is started. However, use of this method to replace the CDT destroys the channel usage statistics kept in the CDT entries. The preferred way to make a channel configuration change take effect immediately is to issue the initializer command, multics, and then with the initializer process in admin mode, modify the CMF, compile it, install the CDT using the install command (which preserves channel usage statistics), and then shut down and reboot Multics.

SAMPLE CHANNEL MASTER FILE

```
/* Sample Channel Master File */

Service:  login;
Charge:   none;
Terminal_type: none;
Line_type: none;
Attributes: none;
Baud: 300;
Access_class: "system_low";
FNP_required_up_time: 2;
Check_acs: priv_attach, dial_out;

FNP:      A;
   type:  DN6670;
   memory: 64;
   hsla:  1;
   image: mcs;

name: a.h000;
   baud: 9600;
   terminal_type: VIP7801;
   attrributes: hardwired, dont_read_answerback;
   initial_command: "modes echoplex,^tabs,tabecho,crecho,lfecho";
   comment: "1200 baud, hsla 0, subchn 0";

/*temporarily disconnected, so comment it out:
name: a.h001;
   baud: 1200;
   charge: caa;
   terminal_type: ASCII;
   comment: "1200 baud, hsla 0, subchn 1, x319, vadic";
   end comment out */
```

```
name: a.h002;
   baud: auto;
   comment: "auto baud, hsla 0, subchn 2, x316";

name: a.h003;
   comment: "300 baud, hsla 0, subchn 3, x324";

Baud: 4800;
name: a.h012;
   service: slave;
   line_type: BSC;
   dataset: 208A;
   comment: "hsla 0, subchn 12, BSC";

name: a.h013;
   service: slave;
   line_type: BSC;
   dataset: 208A;
   comment: "hsla 0, subchn 13, BSC";

name: a.h014;
   baud: 4800;
   line_type: BSC;
   service: multiplexer;
   multiplexer_type: hasp;
   terminal_type: IMFT_HASP_WORKSTATION;
   comment: "HASP multiplexer used for file transfer";

name: a.h014.opr;
   service: slave;
   comment: "IMFT HASP daemon - operator control stream.";

name: a.h014.rdr1-a.h014.rdr8;
   service: slave;
   comment:"IMFT HASP daemon I/O stream.";

name: a.h014.pun1-a.h014.pun4;
   service: slave;
   comment:"IMFT HASP daemon I/O stream.";

name: a.h014.prt1-a.h014.prt4;
   service: slave;
   comment:"IMFT HASP daemon I/O stream.";

name: a.h017;
   charge: caa;
   terminal_type: ASCII;
   attributes: hardwired;
   comment: "4800 baud, hsla 0, subchn 17, hw Delta 4000";
```

```
name: a.h018;
   baud: 1200;

   charge: caa;
   terminal_type: ASCII;
   comment: "1200 baud, hsla 0, subchn 1, x319, vadic";

name: a.h020;
   baud: 2400;
   service: multiplexer;
   multiplexer_type x25;
   line_type: X25LAP;
   terminal_type: X25_TYMNET;
   comment: "TYMNET voice-grade HDLC X25 link.";

name: a.h020.001-a.h020.028;
   initial_command: echo;
   comment: "TYMNET X.25 login subchannels";

name: a.h020.029-a.h020.032;
   service: autocall;
   line_type: X25LAP;
   comment: "TYMNET X.25 autocall subchannels.";

name: a.h030;
   line_type: POLLED_VIP;
   baud: 4800;
   service: multiplexer;

   multiplexer_type:  vip7760;
   comment: "4800 VIP7760 polled controller";
   terminal_type:  VIP7760_CONTROLLER;

name: a.h030.d01;
   terminal_type: VIP7760;
   comment: "Station 01 on polled VIP";

name: a.h030.d02;
   terminal_type: VIP7760;
   comment: "Station 02 on polled VIP";

end;
```

SECTION 5

MODIFYING COMMUNICATIONS CHANNELS

This section describes the recommended procedures for modifying the configuration of communications channels on the Multics system. In particular, it describes the procedures used to add a channel, delete a channel, or change the characteristics or status of an existing channel.

In general, one or two major steps are involved in modifying the channel configuration. The first step is changing the CDT, which can be done in any sufficiently privileged process, as follows:

1. Use a text editor to make the necessary changes to the CMF;
2. Use the cv_cmf command to create a CDT from the new CMF (in >udd>sa>a)
3. Use the install command to install the new CDT.

The syntax of the CMF is described in Section 4, the cv_cmf command is described in Section 7, and the install command is described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No.: GB64.

The second major step, which is not necessary in all cases, is reloading (or reinitializing) the multiplexer of which the channel being changed is a subchannel. In the case of physical channels, of course, this is an FNP (see the discussion of multiplexers and subchannels in Section 1). This is done by means of the load_mpx initializer command, described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No.: GB64, and requires the use of the operator's console, a terminal dialed to the initializer, or the send_admin_command command (described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No.: GB64). It is suggested that the -check control argument to the load_mpx command be used to obtain information about possible inconsistencies in the configuration.

Loading a multiplexer while one or more of its channels is in use is not recommended unless all such users are warned and given a chance to log out. In any case, the load_mpx command, unless given the -force control argument, refuses to proceed if any of the multiplexer's channels are in use. The stop_mbx initializer command can be used to prevent any idle channels from being dialed up to a multiplexer scheduled for reloading.

ADDING CHANNELS

To add a channel to the system, simply add the description of the channel to the CMF, compile and install the new CDT, and, when appropriate, reload the immediately superior multiplexer (e.g., if adding channel a.h007, reload FNP a). Once this has been done, and the appropriate physical connection has been made, the channel is ready for use. Note that the reloading of the multiplexer is not necessary if the channel was configured the last time the multiplexer was loaded; i.e., if two new CDTs have been installed since the last load of the multiplexer, one deleting the channel and the other restoring it, the channel is then ready for immediate use without reloading the multiplexer.

DELETING CHANNELS

To delete a channel from the system, delete (edit) its entry from the CMF, and compile and install the new CDT. If the channel is in use at the time, the installation of the CDT causes it to be hung up and any associated user process to be destroyed. The channel is then made unusable. Note that the multiplexer does not have to be reloaded to accomplish this effect; in fact, if it becomes desirable to restore the channel later, this can be done simply by adding it to the CDT as described above, provided that the multiplexer has not been reloaded in the meantime.

CHANGING THE STATUS OF A CHANNEL

It may be desirable at various times to change the status or characteristics of one or more channels in any of several ways. Changes in the physical configuration may necessitate changing the line type, baud rate, or other attributes of a channel; the site may wish to change the way a channel is used by changing its service type; or it may be necessary to make a channel temporarily unusable. These changes can be effected by changes to the CDT and/or the use of initializer commands; some of them, to take effect, require reloading of a multiplexer. The most common types of changes, and the procedures used to accomplish them, are described below.

Changing Channel Attributes

In general, it is possible to change those attributes of a channel that are specified in the CMF by modifying the CMF entry and compiling and installing the new CDT. The changes thus specified do not, in general, take effect immediately on a channel that is already dialed up, but rather are saved until some change occurs in the state of the channel. The change of state may actually take effect at the next hangup, the next dialup, or the next load of the multiplexer. Table 5-1 shows, for various channel attributes, when a change in the attribute takes effect, depending on whether the channel is currently in use. Some attributes should never be changed on an active channel: the access_class attribute (shown below in Table 5-1) and those entries in Table 5-2 that say "immediate" or "remove".

Attaching, Detaching, and Removing Channels

The initializer commands attach, detach, and remove (described in the *Multics Administration, Maintenance, and Operations Commands* manual, Order No.: GB64) are used to change the state of a channel as perceived by the system. The following paragraphs describe their effects and the circumstances under which they should be used.

attach

The attach command puts a previously inactive channel into a state where it can accept dialups. The channel might be inactive as the result of a detach or remove command (see below), or through some error condition which rendered it inactive. However, a channel defined at CDT installation time with a service type of inactive can be placed in another state only through a subsequent CDT installation.

Table 5-1. Changing Channel Attributes

Attribute	Channel in Use	Channel Not in Use
access_class	dialup (see Note below)	immediately
baud	multiplexer load	multiplexer load
answerback	dialup	dialup
line_type	hangup	immediately
dataset	multiplexer load	multiplexer load
terminal_type	dialup	dialup
initial_command	dialup	dialup
charge	dialup	dialup
set_modes	dialup	dialup
audit	dialup	dialup
hardwired	hangup	dialup
check_answerback	dialup	dialup
dont_read_answerback	dialup	dialup
multiplexer_type	multiplexer load	multiplexer load

NOTE: If the new access_class is lower than the authorization of the process using the channel, the process is bumped.

detach

The detach command renders a channel unusable until it is reattached or its multiplexer is reloaded. It can be used to disable a channel temporarily. If a user is logged in over the channel, the user is bumped as a result of the detach command.

remove

The remove command has the same effect as the detach command. The principal difference is that it forcibly hangs up the channel and detaches it from any user process, whereas the detach command bumps the user by signalling the answering service. The remove command can thus be used in cases when the channel or its associated data bases are in an inconsistent state, and a detach or bump operation has failed to complete properly. Certain changes in service type cause a channel to be automatically removed, as explained below. A removed channel, like a detached one, is restored by the attach command, or when the multiplexer is next loaded.

SERVICE TYPES

If a channel presently being used or available for one type of service (e.g., user login) is to be changed to a different type (e.g., autocall), its CMF entry must be modified and a new CDT compiled and installed. The change of service type may not take effect immediately, particularly if the channel is in use. In some cases, the change does not take effect until either the multiplexer is reloaded or the channel is detached and attached; in other cases, the installation of the CDT automatically removes the channel, and the new service type goes into effect only when it is reattached. Table 5-2 shows, for each combination of old and new service type, when the change actually takes effect. A notation of "attach" indicates that the channel must be detached and reattached; a notation of "remove" indicates that the channel is automatically removed, and must be attached (by operator command, or a subsequent CDT installation or answering service startup) before it can be used with its new service type. Reloading a multiplexer implicitly attaches all of its subchannels whose service types are not "inactive."

Table 5-2. Changing Service Types

Service Type Old\New	login	MC	slave	autocall	multiplexer	inactive
login	--	remove	remove	hangup(1)	remove(2)(3)	remove
MC	attach	--	attach	attach	attach(2)(3)	hangup(1)
slave	immediate	attach	--	immediate	remove(2)(3)	hangup(1)
autocall	hangup(1)	remove	remove	--	remove(2)(3)	remove
multiplexer	remove(4)	remove(4)	remove(4)	remove(4)	--	remove
inactive	immediate	attach	attach	immediate	attach(2)	--

(1) If the channel is not dialed up, the change takes effect immediately.

(2) The multiplexer channel cannot be used until its parent multiplexer (generally an FNP) is reloaded.

(3) A non-multiplexed channel of a running multiplexer cannot be changed to a multiplexed type.

(4) A multiplexed channel of a running multiplexer cannot be changed to a non-multiplexed type.

FNP AND MULTIPLEXER CONFIGURATION

Multiplexers (including FNPs) can be added, deleted, and have their states changed in ways somewhat analogous to those described above for ordinary channels. Adding a multiplexer is like adding any other channel; the channel does not become usable until its parent multiplexer is loaded. In the case of an FNP, there must be a corresponding prph card in the config deck (see the description of the config deck in the *Multics System Maintenance Procedures* manual, Order No.: AM81). In other words, if a new CDT is installed with an additional FNP, and no corresponding prph card appears in the config deck, that FNP cannot be loaded.

Deleting a multiplexer channel implicitly deletes all its subchannels. (In fact, a CMF specifying subchannels of an undefined multiplexer channel is in error and cannot be compiled). Changing the service type of an existing non-multiplexed channel to "multiplexer" is equivalent to adding a new multiplexer channel. As indicated in Table 5-2, such a channel is generally removed when the new CDT is installed, and cannot be used as a multiplexer until its parent multiplexer is next loaded.

The stop_mpx initializer command (described in the the *Multics Administration, Maintenance, and Operations Commands* manual, Order No.: GB64) causes the multiplexer to refuse to accept dialups on its subchannels without affecting any subchannels that are already dialed up. It can be used while the multiplexer is running or before loading it; in the latter case, the multiplexer can be initialized without accepting dialups from any subchannels. In either case, the effect of the stop_mpx command can be reversed by using the start_mpx command.

If a multiplexer disconnects (or, in the case of an FNP, crashes), all its subchannels are automatically hung up. They are not listened to again until the multiplexer is reloaded (if any of the subchannels is itself a multiplexer, the hangup is of course propagated to its subchannels, and so on). The reload is usually performed automatically by the initializer. If, however, a multiplexer crashes repeatedly in a short interval (namely, the number of minutes specified for the FNP_required_up_time statement in the CMF), a message such as:

```
FNP a is in apparent crash loop and will not be reloaded
```

is printed at the initializer terminal, and automatic reloading is discontinued. At this point, it is advisable to consult the programming staff. Once whatever is wrong has been fixed, the multiplexer can be reloaded manually using the load_mpx command.

FNP Crash Notification

It is possible to set up at a given installation an exec_com to be invoked automatically whenever the FNP crashes, which informs the appropriate personnel of the subsequent automatic dump. The exec_com might:

- send mail as notification of the existence of the dump
- automatically format and dprint the dump
- automatically backup to an older core image

The administrator creates an exec_com named fnp_crash_notify.ec in the directory >sc1. The exec_com may contain, for example:

```
&command_line off
sm Prog sm [string "FNP" &1 "crashed at" [date_time] "running" &2]
```

Then, after an FNP dump is taken, the answering service looks for the existence of >sc1>fnp_crash_notify.ec. If this segment is found, it is invoked as follows:

```
ec fnp_crash_notify fnp_tag core_image_path dump_path
```

There is no default exec_com; each installation must prepare its own.

SECTION 6

FNP CORE IMAGES

Programs that run in the FNP are loaded from a "core image" that resides in the Multics storage hierarchy. The pathname of this core image is specified in the CDT, as explained in Section 4.

MODIFYING FNP CORE IMAGES

It may be desirable, on occasion, to modify this core image in order to make use of improvements or bug fixes to the software; to add a module in order to run a new line type; to delete unneeded programs in order to make more buffer space available in the FNP; or to change the maximum number of HMLCs that may be configured. Any such change does not take effect until the FNP is next loaded.

The system administrator does not usually have to change the contents of an FNP module; if he does, however, he should edit the source segment, which is named MODULE.map355, where MODULE is the name of the module. The map355 command (described in Section 7) is then used to produce an object segment named MODULE.objdk, which is used in the next step (binding). Before binding can be performed, however, object segments in the FNP that are unchanged must be extracted from the object archive (see the *Multics Commands and Active Functions* manual, Order No.: AG92) into a directory in the search list (the default is the working directory).

In order to prepare a new core image, the bind_fnp command must be used. If the configuration of HMLCs is being changed, memory size is being changed, or modules are being added or deleted, the bindfile must be modified. The bindfile is a segment named IMAGE.bind_fnp, where IMAGE is the name of the core image to be generated (usually "mcs"). The output of the bind_fnp command is a core image segment named IMAGE. The bind_fnp command is described in Section 7.

USING FNP CORE IMAGE

To make use of the core image thus created, either copy it to the place in the storage hierarchy described by the image pathname in the CDT, or change the specification of the core image pathname in the CMF, convert it, and install the resulting CDT. The next time the FNP is loaded (either automatically or by means of the load_mpx operator command), the new core image is used.

REQUIRED MODULES

The Multics Communication System requires that a number of modules be present in the FNP core image. These modules form the basic software to support the activities of the channels attached to the FNP. The modules are:

```
dia_man
interpreter
scheduler
utilities
init
```

Note that the init module is required in the core image but that after it is finished initializing, its space in the FNP memory is released for use as input or output buffers.

OPTIONAL MODULES

All other modules are optionally configured depending on site requirements. Some of these modules are required because of the line_type statement in the CMF. The following table indicates which module is required in the core image when a channel is configured on an FNP with a line type in the CMF:

line type in CMF	FNP module
ASCII	control_tables
* Sync	control_tables
G115	g115_tables
BSC	bsc_tables
* VIP	vip_tables
POLLED_VIP	polled_vip_tables
X25LAP	x25_tables

The autobaud_tables module is required when the baud rate of a channel is specified as "auto".

*
| The hsla_man module must be in the bindfile for a DN6670 FNP.

The trace module is optional and if left out, the module keyword with its size and mask must also be removed.

The acu_tables module is required in the FNP core image if one of its channels has the service type of "autocall" in the CDT.

The hasp_tables module is required with bsc_tables if a HASP multiplexer channel is configured on the FNP. The line type of this channel is BSC in the CDT, causing bsc_tables to take control of the 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.

The breakpoint_man module is only required if the system maintenance personnel want to set breakpoints in a control tables module. This mechanism is only defined to operate on code that is interpreted by the interpreter module and constructed of op_blocks. *

The ibm3270_tables module is required with the bsc_tables module to service a channel using IBM3270 protocol. The line type of this channel is BSC in the CDT causing bsc_tables to take control of the channel when the FNP is told to listen to it. The ibm3270_ I/O module in the control system indicates to bsc_tables through a line control order that it is to use the ibm3270_tables to help with the line protocol. Refer to the *Multics Subroutines and I/O Modules* manual, Order No.: AG93, for further information.

The console_man module is only required if the console keyword parameter is yes.

The meters module is required if the meter keyword parameter is yes or omitted.

The ic_sampler module is only required if the ic_sample request to the debug_fnp command is to be used. * *

There is no macros module. The macros.map355 is the source that creates the 355_macros library that is used by the map355 command to assemble all the modules that execute in the FNP. The macros.map355 source is also used in creating a data base utilized by the debug_fnp command to reference values in the FNP by name rather than by absolute location.

A GCOS job named macros_asm is used to produce the 355_macros library. If it is necessary to generate a new macro library, this job should be run in a working directory that contains the macros.map355 source, using a command of the form:

```
gcos >ldd>mcs>info>macros_asm -truncate {other -control_args}
```

where other -control_args may include, for example, -list and -lower_case to produce an ASCII list segment. Refer to the *Multics GCOS Environment Simulator* manual, Order No.: AN05, for more information.

The gicb module is kept separate in a suitable form for loading into the FNP in the directory >system_library_unbundled. This is the first executable code loaded into the FNP * which loads the core image into the FNP.

THE BINDFILE

The bindfile is a segment containing symbolic instructions that control the operation of the bind_fnp command. Its entryname ends with a suffix of bind_fnp.

The symbolic instructions of the bindfile have their own syntax that allows for statements consisting of a keyword followed by zero or more parameters, and ending with a statement delimiter. Each keyword designates a certain action to be undertaken by the bind_fnp command pertaining to parameters following the keyword.

Following is a list of the delimiters used:

: keyword delimiter. It is used to identify a keyword followed by one or more parameters. A keyword that is followed by no parameters is delimited by a statement delimiter.

; statement delimiter.

, parameter delimiter (the last parameter is delimited by a statement delimiter).

/* begin comment.

*/ end comment.

Bindfile Key Words

hsla

the parameter is the maximum number of high speed line adapters (HSLA) that this core image supports. Note that in this manual, HMLCs are discussed in terms of HLSAs. Four HMLCs are the equivalent of one HLSA.

*

memory

the parameter is the amount of memory available on the FNP expressed in units of 1024 18-bit words.

console
the parameter can be:

```
yes  console can be configured.
no   console cannot be configured.
```

printer
the parameter can be:

```
yes  printer can be configured.
no   printer cannot be configured.
```

meter
the parameter can be:

```
yes  metering is enabled.
no   metering is disabled.
```

order
the parameters are a list of FNP object segments in the order they are to be loaded. The suffix objdk is assumed for all object segments. The last segment specified must be init.

entry
the parameter specifies the name of the entry at which execution is to begin after the segment is loaded into the FNP. Unless the site is using a changed version of the init module, the entry must be istart.

version
the parameter is a string of up to four characters that is stored in the core image as its version identifier. This string is printed when the FNP is loaded, and is also printed by the tty_meters command.

module
the parameter specifies the name of the FNP object segment that will be described by the following keywords.

type
the parameter specifies the type of FNP object segment. It can be one of the following: hsla or trace.

size
the parameter is the number of 18-bit words required for table space. The use of the size statement depends on the type specified for this module. If the type is trace, then size represents the size of the trace table. If the type is hsla, then the size represents the size of each HSLA table. If a site is configuring less than the maximum number of HSLAs, the bind_fnp command will remove unused table space from the end of the module.

mask

the parameter is a 6-digit octal number that specifies the modules to be traced. Only the six high-order bits (two digits) of this number are interpreted; each bit corresponds to one module. If the bit is "1"b, the corresponding module is traced. The correspondence between bits and modules is as follows:

bit	octal representation	module
0	40XXXX	scheduler
1	20XXXX	dia_man
2	10XXXX	interpreter
3	04XXXX	utilities
* 5	01XXXX	hsla_man

This keyword can only be used if a trace type was also specified.

end;

specifies the end of information to be processed from the bindfile. It must be present at the end of the bindfile.

Sample bind_fnp File

```
/* Bindfile for Multics Communications System */


/*  FNP configuration info */

version:  5.0;

hsla:     3;
*  memory:   64;
console:  no;
meter:    yes;
printer:  no;

/* module load list */

order:    scheduler,
          interpreter,
          control_tables,
          dia_man,
          hsla_man,
          utilities,
          meters,
          trace,
          init;
```

```
/* entry to init from bootload */

entry:    istart;


/* table size specifications */

module:   hsla_man;
 type:    hsla;
 size:    97;

module:   trace;
 type:    trace;
 mask:    310000;    /* trace enable mask */
 size:    512;

end;
```

SECTION 7

COMMANDS

This section contains descriptions of commands useful for channel management and site communications administration.

The conventions shown in the usage lines of these commands are the same as those used throughout the set of Multics manuals; briefly, arguments enclosed in braces ({}) are optional, and all others are required (unless otherwise noted). For a complete description of all the usage line conventions, refer to Section 1 of the *Multics Commands and Active Functions* manual, Order No.: AG92.

Name: bind_fnp

SYNTAX AS A COMMAND

```
bind_fnp pathname {-control_args}
```

FUNCTION

The bind_fnp command produces a core image segment that can be loaded into the FNP. It uses two control segments: a bindfile, which specifies the configuration that the FNP will support, the names and ordering of the object segments included in the core image, and the size of certain software tables; and an optional search rules segment, which specifies which directories are searched to find the object segments.

ARGUMENTS

pathname
: specifies the pathname of the bindfile. If pathname does not have a suffix of bind_fnp, one is assumed.

CONTROL ARGUMENTS

-cross_ref, -cref
: adds a symbol cross reference to the listing segment. If -cross_ref is specified, the listing is generated regardless of whether -list is also specified.

-list, -ls
: produces a listing segment whose name is derived from the name of the bindfile, with the suffix changed to list. The listing segment is a record of the binding. It contains a copy of the bindfile, a load map, and any error messages generated during binding.

-search, -se
: indicates that the user wishes to specify the rules used to search for Multics Communications System object segments being bound into the core image. If given, there must be a segment in the working directory containing an ASCII list of relative pathnames of directories to be searched in the order in which the search is desired. By default, the working directory is searched. This segment must have the same entryname as the bindfile, but with the suffix changed to search.

-version STR
: assigns a version of STR to the core image. The maximum length of STR is four characters. If this control argument is given, it overrides the version keyword specified in the bindfile.

NOTES

A default bindfile is supplied with the system. In general, the only fields that a site administrator would change are: hsla, version, order, and the size keyword for the trace* module.

When creating a new FNP core image, object segments that are unchanged must be extracted from the object archive (see the *Multics Commands and Active Functions* manual, Order No.: AG92) into a directory in the search list before executing the bind_fnp command.

The syntax of the bindfile is described in Section 6.

channel_comm_meters

Name: channel_comm_meters

SYNTAX AS A COMMAND

```
channel_comm_meters channel_name {-control_args}
```

FUNCTION

The channel_comm_meters command prints out metering information for a specified communications channel or channels.

ARGUMENTS

channel_name
is the name of the channel for which information is to be printed. If it is the name of an FNP, totals for that FNP are reported. If channel_name is a starname, information for every channel matching the starname is printed.

CONTROL ARGUMENTS

-brief, -bf
causes a reduced amount of information to be printed for each specified channel.

-error
causes only those meters to be printed that reflect error conditions.

-since_bootload, -boot
prints the meters accumulated since each channel's parent multiplexer (or, in the case of an FNP, the system) was last loaded. This control argument is incompatible with -since_dialup (below).

-since_dialup, -dial
prints the meters accumulated since the channel last dialed up. This is the default. This control argument is incompatible with -since_bootload (above).

-summary, -sum
causes a one-line summary to be printed for each specified channel. This control argument may not be specified if either -brief or -error is specified.

NOTES

If a single channel is specified, the caller must either be the current user of the specified channel or have access to either the metering_gate_ gate or the phcs_ gate. If a starname is specified, the user must have access to one of the above-named gates.

If -brief and -error are both specified, then only those error indications that would be printed with -brief are printed. See the example below.

channel_comm_meters

EXAMPLES

In the example below, code characters appear at the beginning of some lines; these characters do not appear in the actual output of the command. The interpretation of the characters is as follows:

```
A -- this line appears for asynchronous channels only
S -- this line appears for synchronous channels only
B -- this line is among those printed if -brief is specified
E -- this line is among those printed if -error is specified
```

Only lines marked with both B and E are printed if -brief and -error are both specified.

```
channel_comm_meters a.h000

Total metering time 01:45:13

a.h000
```

[The following meters are printed for all nonmultiplexed channels:]

```
                                before conversion   after conversion   ratio
B  Total characters input              984                935            0.95
B  Total characters output          10,540             11,400            1.09
B  Average length of input             8.7                8.3
B  Average length of output           63.1               69.4

                                        read      write     control     total
   Number of calls                       175        194          53       422
   Average time per call (msec.)         2.3        5.8         1.7       4.1
   Average chars. processed per call     5.6       56.1

   Number of software interrupts         113
   Average time per interrupt (msec.)    1.6
   B  Effective speed (bps)              1.6       17.5
```

[The following meters are printed for physical FNP channels only:]

```
SB  Messages transmitted                  240        224
SB  Minimum message length                  5         12
SB  Maximum message length                143        508
SB  Average message length                 10.3       57.6

SBE Invalid input messages              6 (2.5% of total)
SBE Output messages retransmitted       8 (1.6% of total)
SBE Timeout waiting for acknowledge     2 (0.4% of output messages)

    Output overlaps in FNP              127
    Average length of DIA request queue 1.7 entries
```

channel_comm_meters

```
A     Pre-exhaust status                        12
A  E  Exhaust status                             7
A  E  Software transfer timing errors            0
A  E  Bell/quits                                 8
A  E  Echo buffer overflows                      2
   E  Parity errors                              0

      Avg. number of pending status events      1.9
   E  Software status queue overflows           1
   E  Hardware status queue overflows           0
   E  Input buffer allocation failures          1
```

[The following meters are printed for an entire FNP:]

```
      FNP has been up for                  04:15:12
   B  Number of channels configured           88
   B  Average number dialed up               43.7
   B  FNP idle                               74.9%
   B  Idle at peak load                       8.0%

                                  input       output
   B  Characters transmitted  71,966,400  94,934,400
   B  Characters per second        4,700       6,200

    E Abnormal DIA status events                3
    E Memory EDAC errors                        0

   B  Memory size                             64K
   B  Total available buffer pool        6,360 words
   B  Avg. amount of free space         21,876 words
   B  Average % of buffer pool available     34.7
   BE Buffer allocation failures               12
    E Output restricted by space               24


      Number of interrupts from this FNP  1,964,208
      Avg. time/interrupt (ms)                3.1
      % of total CPU time                     1.1
```

```
   Mailbox transactions:
      Input data                              220,349
      Output data                             543,210
      Input control                            14,111
      Output control                           23,456
   ------------------------------------------------------------
      Total                                   801,126

   Average inbound mailboxes in use            1.1
   Average outbound mailboxes in use           3.1
   Maximum outbound mailboxes in use          16
E  No outbound mailbox available              37
E  Input rejects                              22
E  % of input transactions rejected            0.01
```

The following example shows the format of the output of the command when the -summary control argument is specified.

```
channel_comm_meters a.h00* -summary

 cps   cpsi   cpso  iotxXsbepQqa  err  ABE  name        user

 120    0.2    5.4     xX b  Q     12  aB   a.h000      Coren
 600    2.1  102.1    t X      a   73  s    a.h005      ABClone
  30    0.5    2.6         e       2  a E  a.h009      Parrish
```

The column headings are interpreted as follows:

cps
the nominal speed of the channel, in characters per second.

cpsi
the effective speed of input over the channel, in characters per second.

cpso
the effective speed of output over the channel, in characters per second.

channel_comm_meters

The following flags are printed if the corresponding condition has occurred at least once on the channel.

```
i -- invalid input message
o -- output message retransmitted
t -- timeout waiting for acknowledge
x -- pre-exhaust status
X -- exhaust status
s -- software transfer timing error
b -- bell/quit
e -- echo buffer overflow
p -- parity error
Q -- software status queue overflow
q -- hardware status queue overflow
a -- input buffer allocation failure
```

err
: the total number of errors of all kinds that have occurred on the channel.

A
: "a" for an asynchronous channel or "s" for a synchronous channel.

B
: the channel is in breakall mode.

E
: the channel is in echoplex mode.

name
: the name of the channel.

user
: the Personid of the current user of the channel. If the channel is not in use, or the user's name is not available, this field is left blank.

Name: console_report

SYNTAX AS A COMMAND

```
console_report {as_log_paths} {-control_args}
```

FUNCTION

The console_report command creates and displays metering of terminal usage on the system based on the information obtained from the answering service logs.

ARGUMENTS

as_log_paths
are pathnames of answering service logs. The pathnames can be absolute or relative.

CONTROL ARGUMENTS

-print
causes a display of the metering on user_output.

-report_reset, -rr
causes a display of the metering on user_output and resets the metering in the data base.

-reset, -rs
resets the metering in the data base.

-sort
sorts the data base alphanumerically by terminal answerback identifiers.

NOTES

Control arguments and pathnames can appear anywhere in the command line. They are processed from left to right, one at a time.

The header in the display produced when the -print and -report_reset control arguments are used is obtained from the system titles. This header may be too wide for the user's terminal and the user may wish to use the file_output command.

The command creates and stores data into two segments, termseg and termuseg, in the current working directory when pathnames of answering-service logs are specified. These segments are the data bases that the command expects to find in the current working directory when any of the control arguments are used.

console_report

EXAMPLES

```
Multics terminal usage

Period from 08/26/80 1407.9 to 09/25/80 1600.2
    Type      Count    Logins   Nologins       CPU    Connect


    2741        247        28         17      0:05      25.22
    TN300       395       632         48     18:47     604:18
    Daemon      101       873          4     38:08    4456:06
    ASCII      1077      9563       1423    531:45    7954:30
    TELNET       13         0        324      0:00       0:00
    LA36         93        60         11      0:34      15:40
    VT100        41       419         45      6:10     367:53
    LA120         5       112          9      2:24      89:51

ID   Type    Location     Logins     Nologins   CPU      Connect
       User

002  ASCII                  1          3       0:04       3:03
       Derek.Multics        1                  0:04       3:03

004  ASCII                  2          4       0:01       1:16
       Aulin.Multics        2                  0:01       1:16

405  LA120                 73         26       0:46      75:54
       MKane.Network       38                  0:31      53:07
       Marcus.Network       8                  0:04       6:09
       Inada.Network       11                  0:07       5:53
       SBWeber.Network      9                  0:05       6:37
       Ducot.FlexMan        5                  0:01       3:43
       Yip.Network          1                  0:01       0:02
       Feinstein.Network    1                  0:01       0:27

40>  ASCII                  2          0       0:01       0:26
       Stanzel.ARCS         1                  0:01       0:03
       AWhite.ARCS          1                  0:01       0:24

40>  VT100                  2          0       0:01       0:38
       HSPP-BASIC.Student   1                  0:01       0:16
       Soley.ARCS           1                  0:01       0:23
etc.
```

console_report

where:

Type
is the terminal type.

Count
is the number of different terminals of that type.

Logins
are the number of completed logins.

Nologins
the number of logins attempted and not completed.

CPU
is the amount of CPU time used.

Connect
is the amount of time actually logged in.

ID
is an identification number assigned to a specific terminal.

Type User
the terminal type and ID is shown on a heading, followed by a line for each user of the terminal giving usage statistics.

Location
is not used.

cv_cmf

Name: cv_cmf

SYNTAX AS A COMMAND

```
cv_cmf cmf_path {-control_args}
```

FUNCTION

The cv_cmf command converts an ASCII channel master file (CMF) into a binary channel definition table (CDT). The binary table can be installed using the install command.

ARGUMENTS

cmf_path
is the pathname of the channel master file. If path does not have a suffix of cmf, one is assumed. However, the suffix cmf must be the last component of the name of the source segment.

CONTROL ARGUMENTS

-brief, -bf
uses short form of error messages.

-long, -lg
uses long form of error messages.

-severity N, -sv N
causes error messages whose severity is less than N (where N is 0, 1, 2, 3, or 4) not to be written to the user_output switch. If this control argument is not specified, a severity level of 0 is assumed (i.e., all error messages are written to the user_output switch).

NOTES

If no control arguments are given, each error message is printed in long form the first time it occurs and in short form thereafter.

The converted channel master file is given a name corresponding to the entryname of the source segment, with the cmf suffix replaced by cdt. It is placed in the working directory.

NOTES ON SEVERITY VALUES

The cv_cmf command associates the following severity values to be used by the severity active function:

Value	Meaning
0	No compilation yet or no error.
1	Warning.
2	Correctable error.
3	Fatal error.
4	Unrecoverable error.
5	Could not find source.

display_cdt

Name: display_cdt

SYNTAX AS A COMMAND

```
display_cdt {channel} {-control_args}
```

FUNCTION

The display_cdt command enables a qualified user to display the contents of a channel definition table (CDT).

ARGUMENTS

channel
is the name of the communications channel for which the CDT entry is to be displayed. The star convention is allowed.

CONTROL ARGUMENTS

-all, -a
displays names and CDT indices for all channels in the CDT.

-brief, -bf
displays only channel names and CDT indices (without channel or FNP details). This is the default for the -all and -subtree control arguments.

-cmf path
creates a CMF in the segment named path in a form suitable to cv_cmf, based on the contents of the CDT.

-header, -he
displays the CDT header variables in addition to other requested information.

-long, -lg
displays detailed information for the specified channel or FNP. This is the default unless -all or -subtree is specified, in which case -brief is the default.

-no_header, -nhe
suppresses display of the CDT header variables. This is the default.

-pathname path, -pn path
displays the CDT whose pathname is path. By default, the CDT in the segment >sc1>cdt is displayed.

-subtree
displays the names and CDT indices for all subchannels (if any) of the specified channel.

NOTES

If display_cdt is specified with no channel name and no control arguments, a usage error notification is returned. Specifying channel name only, with no control arguments, results in a -long display.

The display_cdt command enables the user to check for inconsistencies in a CDT before unnecessarily undertaking corrective action.

The user must have r access to the CDT to invoke the display_cdt command.

EXAMPLES

To display the entire system CDT, specify:

```
display_cdt -all -header
```

To display the system CDT entry for channel a.1000 only, specify:

```
display_cdt a.1000
```

To display all the subchannels of multiplexer a.h026.1, specify:

```
display_cdt a.h026.1 -subtree -long
```

display_fnp_idle

Name: display_fnp_idle

SYNTAX AS A COMMAND

```
display_fnp_idle {fnp_names} {-control-args}
```

FUNCTION

The display_fnp_idle command is used to display information on FNP idle time stored by the meter_fnp_idle command. The display can be in the form of a summary or a line graph (histogram).

ARGUMENTS

fn_names
are the names of the FNPs for which idle time information is to be displayed. If fnp_names is not specified, the display covers all FNPs for which information has been stored.

CONTROL ARGUMENTS

-directory path, -dr path
specifies that information is to be taken from segments in the directory with pathname path (see the meter_fnp_idle command description for idle time segment specifications). The default is to display information from idle time segments in the working directory.

-from DT, -fm DT
specifies that the display is to cover a period beginning no earlier than the date/time DT, which must be in a form suitable for input to the convert_date_to_binary_ subroutine. The default is to start the display from the most recent idle time segment.

-histogram, -hist
causes output in the form of a histogram, where a line shows the busy percentage for each FNP at a given time interval. The -histogram and -summary control arguments are mutually exclusive, but one or the other must be specified.

-interval N
specifies that each line in the histogram represents an N minute interval. This control argument is ignored if -summary is specified. The default is 15 minute intervals.

-line_length N, -ll N
specifies the line length of the histogram as N columns (N cannot be less than 38). This control argument is ignored if -summary is specified. The default is the user's terminal line length (or 80 if output is directed to a file).

-summary, -sum
requests a summary display of FNP idle information for the specified time period. The -summary and -histogram control arguments are mutually exclusive, but one or the other must be specified.

-to DT
specifies that the display is to cover a period ending no later than the date/time DT, which must be in a form suitable for input to the convert_date_to_binary_ subroutine. The default is to end the display with the latest available information.

EXAMPLES

The following command and resultant display represent a sample display_fnp_idle summary request in which FNP "a" was found to be 90.3% idle for the specified period (i.e., starting from the beginning of the most recent idle time segment and including the latest available information). Note that the busiest interval within the period is identified.

```
display_fnp_idle a -summary

FNP A idle from 01/04/82 0600. to 01/06/82 1436.: 90.3%
Busiest sample interval:
01/05/82 1423. to 1424.: 43.7% idle
```

The following command and resultant display represent a sample display_fnp_idle histogram request for all FNPs for which idle time has been recorded. The time period starts no earlier than noon on 01/06/82 and includes the latest available information. The busy percentage for each FNP is given at 10-minute intervals.

```
display_fnp_idle -from "01/06/82 1200." -hist -interval 10

          %busy  0      10      20      30      40      50      60
                 |      |       |       |       |       |       |
01/06/82  1330.     A     B  C
          1340.          A   C    B
          1350.  A             CB
          1400.  A              B
          1410.     A          C
          1420.          A    B  C
          1430.            A B                                  C
```

fnp_throughput

Name: fnp_throughput

SYNTAX AS A COMMAND

```
fnp_throughput {fnp_id} {-control_arg}
```

FUNCTION

The fnp_throughput command reports character throughput for an FNP or all FNPs and optionally allows the resetting of the metering interval.

ARGUMENTS

fnp_id
is the name of an FNP or "*", which means all currently running FNPs. This argument must be specified unless the -reset control argument is specified, in which case fnp_id must not be specified.

CONTROL ARGUMENTS

may be either, but not both, of the following. If neither is specified, information is printed and the metering interval is not reset.

-report_reset, rr
causes statistics to be printed and the metering interval to be reset. If this control argument is specified, fnp_id must be specified.

-reset, -rs
causes the metering interval to be reset without printing any statistics. If this control argument is specified, fnp_id must be omitted.

NOTES

The start of the metering interval in effect is measured from the time the FNP was last booted, or from the time the interval was last reset, whichever was the most recent event.

The reset action of the -report_reset control argument applies to the metering interval for all FNPs, even though the command invocation is specific to the statistics for a particular FNP.

EXAMPLES

To report on throughput statistics for FNP "a", while resetting the metering interval, specify:

```
fnp_throughput a -report_reset
```

The output would appear as follows:

```
                          input          output
Characters transmitted   71,966,400     94,934,400
Characters per second         4,700          6,200
```

Name: map355

SYNTAX AS A COMMAND

```
map355 pathname {-control_args}
```

FUNCTION

The map355 command is used to assemble a program written in the FNP assembler language, map355. The command does not assemble the program directly. Instead, it prepares a GCOS job deck to perform the assembly and calls the GCOS Environment Simulator to do the work.

ARGUMENTS

pathname
is the pathname of the source program to be assembled. The suffix of map355 need not be given by the user; it is assumed.

CONTROL ARGUMENTS

-list, -ls
creates a listing segment that documents the compilation. The listing is

-macro_file path
specifies the pathname of the macro file to be used for the assembly. If omitted, >ldd>mcs>info>355_macros is used.

-check
specifies that the compiler only perform a syntax check of the source. No object segment is created.

-comdk
creates a GCOS comdk segment. This segment contains a BCD version of the source program. It is created in the working directory with a suffix of comdk.

-gcos_list, -gcls
creates a GCOS listing segment in the working directory. This is a BCD version of the listing segment. It has a suffix of glist.

-noconvert
specifies that the input segment is a GCOS comdk, rather than an ASCII segment. If this control argument is used, the source segment must have a suffix of comdk.

-argument list, -ag list
specifies a list of arguments to be passed to the GCOS Environment Simulator.

NOTES

This command creates a series of segments for use by the GCOS simulator. Some are created in the working directory, some are created in the process directory, and some through links in the working directory to the process directory. These segments and links are normally deleted when the command terminates, leaving just the object segment, which has a suffix of objdk.

Refer to the *GCOS Environment Simulator* manual, Order No. AN05, for more information on the use of the GCOS Environment Simulator.

WARNING

The map355 command creates links in the working directory to segments to be placed in the process directory. If the process terminates in the middle of a compilation (new_proc or a crash), these links will remain. This means that the next time the command is invoked, it will fail because the links point to a nonexistent directory. Even though the command fails, the bad links will be unlinked and subsequent invocations will work correctly.

SEVERITY

The map355 command sets the following severity values to be returned by the severity active function when the "map355" keyword is used:

Value	Meaning
0	No error (or command not yet used)
1	The assembly produced warning flags
2	The objdk segment could not be created

Name: mcs_version

SYNTAX AS A COMMAND

```
mcs_version {fnp_tag}
```

FUNCTION

The mcs_version command prints the version of the core image most recently loaded into the specified FNP.

SYNTAX AS AN ACTIVE FUNCTION

```
[mcs_version {fnp_tag}]
```

where fnp_tag is the identifier of the FNP whose version is to be printed. It can be a, b, c, d, e, f, g, or h. If it is omitted, a is assumed.

meter_fnp_idle

Name: meter_fnp_idle

SYNTAX AS A COMMAND

```
meter_fnp_idle fnp_name {-control-args}
```

FUNCTION

The meter_fnp_idle command reads FNP idle metering information at specified intervals and stores the information in a segment for later viewing through the display_fnp_idle command.

ARGUMENTS

fnp_name
identifies the FNP for which idle time is to be recorded.

CONTROL ARGUMENTS

-directory path, -dr path
specifies that the segments in which idle time information is to be stored are to be created in the directory with pathname path. The default is the user's working directory. Segment names are of the form fnp_idle_data.F.MMDDYY.HHMMSS.I where F is fnp_name, MMDDYY.HHMMSS is the starting date and time of recording, and I is the specified interval in minutes.

-interval N
specifies that metering information is to be sampled every N minutes. The default is 1.

-stop, -sp
terminates meter reading on this FNP. No other control argument may be specified with -stop.

NOTES

Information is appended to the idle time segment until the -stop control argument is encountered or the process terminates. Maximum length of an idle time segment is 64K (to avoid exhausting a 256K ASTE). If a segment becomes full, a new one is created. On average, taking readings at one-minute intervals would fill a 64K segment in a little over three weeks (if a single process in which the command was invoked were to run that long).

Each invocation of the command creates a new segment.

Use of this command requires access either to the metering_gate gate or the phcs_ gate.

The command:

```
meter_fnp_idle a -interval 5
```

creates a segment in the working directory into which will be stored idle time meter readings taken on FNP "a" at five-minute intervals.

set_x25_packet_threshold

Name: set_x25_packet_threshold

SYNTAX AS A COMMAND

```
set_x25_packet_threshold channel_name packet_size
```

FUNCTION

The set_x25_packet_threshold command sets the minimum size of X.25 "long" packets. Packets of this size or larger are given lower priority than short packets.

ARGUMENTS

channel_name
is the name of an X.25 multiplexer channel.

packet_size
is the minimum length of a long packet, in characters.

NOTES

If packet_size is set larger than the maximum packet size in use by the multiplexer, no packets are given preferential treatment on the basis of size. The initial value of the minimum long packet size is determined by the "packet_threshold" parameter in the terminal type definition for the multiplexer. The present value of the parameter, and the maximum size for all packets, is included in the output from the tty_dump command.

Use of this command requires access to the hphcs_ gate.

Name: system_comm_meters

SYNTAX AS A COMMAND

```
system_comm_meters {-control_args}
```

FUNCTION

The system_comm_meters command prints out metering information for ring zero Multics Communications Management.

CONTROL ARGUMENTS

-reset, -rs
: resets the metering interval for the invoking process so that the interval begins at the last call with -reset specified. The metering information is not printed. If -reset has never been given in a process the interval begins at system initialization time.

-report_reset, -rr
: prints metering information and then resets the metering interval.

NOTES

Use of the system_comm_meters command requires access to the metering_gate_ and phcs_ gates.

EXAMPLES

The following is a sample of the output of the system_comm_meters command.

```
Total metering time 05:43:27

THROUGHPUT
                            before conversion  after conversion  ratios
Total characters input            17,234,567        15,543,210    0.90
Total characters output          168,012,345       185,876,543    1.14
Average length of input             12.3 characters
Average length of output            59.7 characters
Input characters preconverted       20,435 (1.2% of total)

                              read              write
Number of calls               1,456,789         26,357,924
Average time per call          6.37 msec.        9.63 msec.
Average chars. processed      13.5              57.8
Average chars. per msec.       2.1               5.8

CHANNEL INTERRUPTS
                          input          output          other          total
software "interrupts"   678,901         423,440        110,011      1,212,352
average time (msec.)       1.34            0.56           0.23           1.01
```

system_comm_meters

TTY_BUF SPACE MANAGEMENT

Total size of buffer pool	11,480 words
Number of channels configured	143
Number of multiplexed channels	8

% of buffer pool in use	current	average
input	6.9	6.5
output	13.4	15.6
control structures	15.8	15.3
total	36.1	37.4

Smallest amount of free space ever 4,358 words (38% of buffer pool)

	allocate	free	total
Number of calls	24,657,988	20,665,443	45,323,431
Average time per call (msec.)	0.23	0.37	0.29
% of total CPU	0.14	0.17	0.31

Calls requiring loop on tty_buf lock	1,249,340	(2.83% of total)
Average time spent looping on lock	0.14 msec.	(0.01% of total CPU)
Number of allocation failures	0	(0.00% of attempts)

CHANNEL LOCK CONTENTION

Number of calls to tty_lock	40,392,817	
Times channel lock found locked	2,364,758	(5% of attempts)
Average time spent waiting for lock	1.8 msec.	
Maximum time spent waiting for lock	3.7 msec.	
Number of interrupts queued because channel locked	25,437	(2.2% of interrupts)

ECHO NEGOTIATION

Average time of transaction	3.2 msec.	
Number of characters echoed by supervisor	21,576	(0.13% of input chars)
Number of characters echoed by FNPs	335,466	(1.87% of input chars)

ABNORMAL EVENTS

Input restarts	12,576	(0.8% of read calls)
Output restarts	304,289	(1.2% of write calls)
Output space overflows	16,384	(0.1% of write calls)
"needs_space" calls	0	

Name: tty_dump

SYNTAX AS A COMMAND

```
tty_dump channel_name {-control_args}
```

FUNCTION

The tty_dump command displays on the user's terminal the contents of the ring zero data bases describing either the current state of selected communications channels managed by the Multics Communication System or the state of such channels at the time of a system crash. *

ARGUMENTS

channel_name
specifies the communications channels for which the state is to be displayed. The star convention is allowed (e.g., b.h202.**). This argument is incompatible with the -user control argument.

CONTROL ARGUMENTS

-user STR
specifies that the state of all communications channels attached by the specified user(s) is to be displayed. STR is a starname used to identify the users and is matched against the Person_id.Project_id of each logged in user. For example, "*Smith.M*" would match any user whose Person_id ends with "Smith" that is logged in on a project that starts with "M". This control argument is incompatible with the channel_name argument and the -erf control argument.

-erf N
specifies that information about the channels is to be taken from the system dump associated with error report form (ERF) N located in >dumps. If this control argument is omitted, information about the currently running system is displayed. This control argument is incompatible with the -user control argument.

-all, -a
specifies that, for each channel selected by the above arguments, information is to be displayed from the data bases of the channel, its parent multiplexer, its grandparent multiplexer, etc., up to the top level multiplexer channel. For example, for b.h202.prt1, all information from the data bases of b.h202.prt1, b.h202, and b that is related to b.h202.prt1 would be displayed.

-lcte
specifies that the logical channel table entries (LTE) for the selected channels are to be displayed in addition to the other information normally displayed. If -all is specified, the LCTEs of all parent multiplexers are also displayed.

-subchan, -sbc
specifies that information from the data base of the parent multiplexer related only to the selected channels is to be displayed.

-long, -lg
specifies that the contents of any input and output buffers for the channels are to be displayed. This is the default.

-brief, bf
suppresses display of the buffer contents. Only the addresses, size, and flags for each buffer are displayed.

-ascii
specifies that the contents of buffers are to be interpreted as ASCII characters in addition to being displayed as octal or hexadecimal values.

-ebcdic8
specifies that the contents of buffers are to be interpreted as EBCDIC (8-bit byte) characters in addition to being displayed as octal or hexadecimal values.

-ebcdic9
specifies that the contents of buffers are to be interpreted as EBCDIC (9-bit byte) characters, in addition to being displayed as octal or hexadecimal values.

-octal
specifies that the contents of buffers are to be displayed as octal values in addition to any character interpretation. Octal is the default numeric mode for buffer contents display.

-hex8
specifies that the contents of buffers are to be displayed as hexadecimal values, in addition to any character interpretation. Each 8-bit byte in a word is displayed (nine hexadecimal digits).

-hex9
specifies that the contents of buffers are to be displayed as hexadecimal values, in addition to any character interpretation. The low order 8 bits of each 9-bit byte in a word is displayed as two hexadecimal digits.

NOTES

Use of the tty_dump command without the -erf control argument requires access to the gate phcs_.

The description of the dump_segment command in the *Multics Commands and Active Functions* manual, provides detailed information on the various buffer display formats.

The default mode for buffer displays is to display their contents as octal values without any character interpretation.

There are two sets of conflicting control arguments in tty_dump: three with which to specify the base of numeric display (-octal, -hex8, and -hex9), and three with which to specify character code interpretation (-ascii, -ebcdic8, and -ebcdic9). If conflicting control arguments are given on the command line, the last one specified will be used.

EXAMPLES

The command line:

```
tty_dump b.h202.** -all
```

displays the state of multiplexer b.h202 and its subchannels. Displayed information includes the WTCBs/TCBs of the subchannels, multiplexer-specific data for the subchannels, global data for the multiplexer, and the PCB of the multiplexer's physical FNP channel.

By comparison, the command line:

```
tty_dump b.h202 -all
```

displays only global multiplexer data and the PCB.

tty_lines

Name: tty_lines

SYNTAX AS A COMMAND

```
tty_lines {arg} {-control_args}
```

FUNCTION

The tty_lines command prints information about communications channels defined in the channel definition table (CDT). An optional argument can be used to print information about a subset of channels.

arg is either a channel name, in which case information about the specified channel is printed, or one of the following:

ARGUMENTS

id STR
: prints information about channels on which the terminal most recently dialed up has an identification code specified by STR. The string STR is four characters long.

dlN
: prints information about each channel that has been dialed up N times or more.

d=N
: prints information about each channel that has been dialed up exactly N times.

stN
: prints information about each channel whose current state code is N (see item 4 in "Notes" below).

acN
: prints information about each channel whose activity code is N (see item 6 in "Notes" below).

slN
: prints information about the Nth entry in the CDT.

CONTROL ARGUMENTS

-type STR
prints information about each channel on which the most recently dialed inal was of the terminal type specified by STR.

NOTES

If the tty_lines command is given with no argument, it prints information about all channels in the CDT.

For each channel, a line is printed in the following format:

```
NAME   TYPE D S  W A  BAUD Person_id Project_id (ID) C
```

NAME
is the channel name, e.g., a.1006.

TYPE
is the terminal type that has most recently dialed the channel, or NU if the channel has not been used.

D
is the number of times the channel has been dialed up.

S
is the current state of the channel. It may have one of the following values:

```
 1   hung up
 2   listening (ready for dialup)
 5   dialed
-1  masked for excessive interrupts
```

W
is an internal variable indicating what the answering service expects to happen next to the channel.

A
is the activity code for the channel. It may have one of the following values:

```
1   hung up
2   listening (ready for dialup)
3   dialed up but not logged in
4   user is logged in but process not yet created
5   user process has channel
6   auto_call line is in process of dialing out
7   auto_call line is in use (dial complete)
```

tty_lines

BAUD
is the baud rate of the channel

Person_id
is the Person_id of the current user of the channel. If A is not 4 or 5, this field is omitted.

Project_id
is the Project_id of the current user of the channel. If A is not 4 or 5, this field is omitted.

ID
is the identification of the terminal currently using the channel. If S is not 5, this field is omitted.

C
is the comment field from the CDT entry.

SECTION 8

SUBROUTINES

This section contains descriptions of subroutines useful for channel management and site communications administration.

The conventions shown in the usage lines of these subroutines are the same as those used throughout the set of Multics manuals.

Name: comm_meters_

The comm_meters_ subroutine, given a list of communications channel names, returns metering information for all the specified channels. The exact information returned for each channel varies depending on the line type and multiplexer type of the channel. Callers of comm_meters_ should later call the comm_meters_$free entry point to release the space allocated for the returned metering information.

USAGE

```
declare comm_meters_ entry ((*) char (32), fixed bin, pointer, fixed bin,
      pointer, fixed bin (35));

call comm_meters_ (chan_names, version, area_ptr, n_channels, chan_meters_ptr,
      code);
```

ARGUMENTS

chan_names
is an array of channel names, any of which may be starnames. (Input)

version
is the version number of the channel_meters structure to be returned. (Input) It must be 1.

area_ptr
is a pointer to an area in which the returned metering information is to be allocated. (Input) It must be a nonnull pointer.

n_channels
is the number of channels for which metering information is returned. (Output)

chan_meters_ptr
is a pointer to a linked list of structures containing the returned metering information. (Output)

code
is a standard system status code. (Output)

The structure pointed to by chan_meters_ptr has the following format, which is defined in the include file channel_meters.incl.pl1:

```
dcl 1 channel_meters aligned based (chan_meterp),
    2 version fixed bin,
    2 multiplexer_type fixed bin,
    2 parent_type fixed bin,
    2 line_type fixed bin,
    2 flags,
      3 reserved bit (36) unaligned,
    2 pad1 fixed bin,
    2 channel_name char (32),
    2 mpx_specific_meterp pointer,
    2 parent_meterp pointer,
    2 next_channelp pointer,
    2 cumulative,
      3 unconverted_input_chars fixed bin (35),
      3 converted_output_chars fixed bin (35),
      3 read_calls fixed bin
      3 write_calls fixed bin,
      3 control_calls fixed bin,
      3 software_interrupts fixed bin,
      3 read_call_time fixed bin (71),
      3 write_call_time fixed bin (71),
      3 control_call_time fixed bin (71),
      3 interrupt_time fixed bin (71),
      3 pad (4) fixed bin,
    2 saved like channel_meters.cumulative;
```

STRUCTURE ELEMENTS

version
contains the value of the version argument (above).

multiplexer_type
is the multiplexer type of the channel. It may have any of the values defined in multiplexer_types.incl.pl1.

parent_type
is the multiplexer type of the channel's parent multiplexer. If the channel is a level-1 multiplexer (i.e., top-level multiplexer), parent_type is set to -1.

line_type
is the line type of the channel. It may have any of the values defined in line_types.incl.pl1.

flags
are reserved for future use.

channel_name
is the name of the channel.

mpx_specific_meterp
is a pointer to additional meters that vary according to multiplexer type.

parent_meterp
is a pointer to additional meters maintained on behalf of the channel by its parent multiplexer. If the channel is a level-1 multiplexer, this pointer is null.

next_channelp
is a pointer to the channel_meters structure for the next channel in the list. If this is the last channel, next_channelp is null.

cumulative
contains meters for the channel accumulated since the most recent bootload of the system.

unconverted_input_chars
is the number of characters input on the channel before conversion at the channel's multiplexing level.

converted_output_chars
is the number of characters output on the channel after conversion.

read_calls
is the number of calls to channel_manager$read for this channel.

write_calls
is the number of calls to channel_manager$write for this channel.

control_calls
is the number of calls to channel_manager$control for this channel.

software_interrupts
is the number of calls to channel_manager$interrupt for this channel.

read_call_time
is the amount of time (in microseconds) spent in read calls.

write_call_time
is the amount of time spent in write calls.

control_call_time
is the amount of time spent in control calls.

interrupt_time
is the amount of time spent processing software interrupts.

saved
contains meters saved the last time the channel dialed up. They can be used in combination with the cumulative meters above to derive statistics for the current dialup session.

NOTES

If comm_meters_ encounters an error, it calls the sub_err_ subroutine, raising the sub_error_ condition with the restart flag on. The structure pointed to by sub_err_info.info_ptr has the following format, which is defined in the include file comm_meters_error_info.incl.pl1:

```
dcl 1 comm_meters_error_info aligned based (comm_meters_errp),
    2 version fixed bin,
    2 chan_name char (32),
    2 flags,
      3 starname_matched bit (1) unal,
      3 more_than_one_starname bit (1) unal,
      3 more_than_one_match bit (1) unal,
      3 pad bit (33) unal;
```

STRUCTURE ELEMENTS

version
is the version number of the structure. It is set to comm_meters_err_v1.

chan_name
is the name of the channel being processed when the error was encountered.

starname_matched
"0"b error was encountered while matching a starname
"1"b a starname provided by the caller was matched

more_than_one_starname
"0"b the caller provided no more than one starname to chan_names
"1"b more than one starname was provided

more_than_one_match
"1"b starname currently being processed was matched by more than one channel name
Meaningless if starname_matched (above) is not "1"b

Entry: comm_meters_$free

This entry is called to release space allocated by comm_meters_ to return metering information. Any program that calls comm_meters_ should subsequently call comm_meters_$free to release the allocated space.

USAGE

```
declare comm_meters_$free entry (pointer, pointer, fixed bin (35));

call comm_meters_$free (area_ptr, chan_meters_ptr, code);
```

ARGUMENTS

area_ptr
is a pointer to the area in which the space was allocated. (Input)

chan_meters_ptr
is a pointer to the list of metering structures returned by comm_meters_ (above). (Input)

code
is a standard system status code. (Output)

Entry: comm_meters_$get_comm_meters

This entry is identical in function to phcs_$get_comm_meters; it exists for the use of callers who lack access to the phcs_ gate. The arguments are the same as for phcs_$get_comm_meters.

Name: MPX_meters_

This is a generic name for a collection of utility subroutines for managing meters specific to various multiplexer types. The string "MPX" in the name represents the name of a multiplexer type, as defined in multiplexer_types.incl.pl1; thus, the actual subroutines have names such as tty_meters_, mcs_meters_, vip7760_meters_, etc.

Entry: MPX_meters_$allocate_mpx

The allocate_mpx entry allocates a structure for meters maintained by a multiplexer for channels of that multiplexer type; e.g., vip7760_meters_$allocate_mpx allocates the metering structure for a vip7760 channel itself.

USAGE

```
declare MPX_meters_$allocate_mpx entry (pointer, pointer, fixed bin (35));

call MPX_meters_$allocate_mpx (area_ptr, meter_ptr, code);
```

ARGUMENTS

area_ptr
is a pointer to an area in which the metering structure is to be allocated. (Input)

meter_ptr
is a pointer to the allocated metering structure, whose format depends on the multiplexer type. (Output) If the entry is being called in preparation for a call to metering_gate_$get_comm_meters, the value of this pointer should be assigned to get_comm_meters_info.subchan_ptr.

code
is a standard system status code. (Output)

Entry: MPX_meters_$allocate_subchan

This entry allocates a structure for meters maintained by the multiplexer for its subchannels; e.g., vip7760_meters_$allocate_subchan allocates space for meters kept by the vip7760 multiplexer modules for subchannels of a vip7760 channel.

USAGE

```
declare MPX_meters_$allocate_subchan entry (pointer, pointer, fixed bin (35));

call MPX_meters_$allocate_subchan (area_ptr, meter_ptr, code);
```

ARGUMENTS

are the same as for the allocate_mpx entry, above. If the entry is called in preparation for a call to metering_gate_$get_comm_meters, the value of meter_ptr should be assigned to get_comm_meters_info.parent_ptr.

Entry: MPX_meters_$free_mpx

This entry is used to free the storage allocated by the MPX_meters_$allocate_mpx entry, above.

USAGE

```
declare MPX_meters_$free_mpx entry (pointer, fixed bin (35));

call MPX_meters_$free_mpx (meter_ptr, code);
```

ARGUMENTS

meter_ptr
    is a pointer to the metering structure to be freed. (Input)

code
    is a standard system status code. (Output)

Entry: MPX_meters_$free_subchan

This entry is used to free the storage allocated by the MPX_meters_$allocate_subchan entry, above.

USAGE

```
declare MPX_meters_$free_subchan entry (pointer, fixed bin (35));

call MPX_meters_$free_subchan (meter_ptr, code);
```

ARGUMENTS

are the same as for the free_mpx entry, above.

Entry: MPX_meters_$display_mpx

This entry displays, on a specified I/O switch, the meters associated with a multiplexer of the specified type. The format of the statistics displayed depends on the type of multiplexer. These entries are called by commands that display general communications meters.

USAGE

```
declare MPX_meters_$display_mpx entry (char (*), pointer, pointer, bit (36)
      aligned, fixed bin (35));

call MPX_meters_$display_mpx (chan_name, iocb_ptr, meter_ptr, flags, code);
```

ARGUMENTS

chan_name
: is the name of the multiplexer channel for which statistics are to be displayed. (Input)

iocb_ptr
: is a pointer to the I/O control block for the I/O switch on which the meters are to be displayed. (Input) If it is null, the user_output switch is used.

meter_ptr
: is a pointer to the structure returned by the comm_meters_ subroutine for the channel specified by chan_name (above). (Input)

MPX_meters_

flags
indicates the control arguments specified to the channel_comm_meters command to control the format of the output. (Input) The individual flags are defined as follows:

```
bit 1      - "1"b if  -brief specified
bit 2      - "1"b if  -error specified
bit 3      - "1"b if  -summary specified
bit 4      - "1"b if  -since_bootload specified
bits 5-36 - reserved
```

These values are defined in the include file comm_meters_disp_flags.incl.pl1.

code
is a standard system status code. (Output)

Entry: MPX_meters_$display_subchan

This entry displays, on a specified I/O switch, meters kept by a multiplexer of the specified type on behalf of one of its subchannels. The format of the displayed meters depends on the multiplexer type.

USAGE

```
declare MPX_meters_$display_subchan entry (char (*), pointer, pointer, bit
      (36) aligned, fixed bin (35));

call MPX_meters_$display_subchan  (chan_name,  iocb_ptr,  meter_ptr,  flags,
      code);
```

ARGUMENTS

are the same as for the display_mpx entry, above.

Name: metering_gate_$comm_chan_star_list

This entry returns a list of communications channel names that match a given starname, along with information on each channel's multiplexer type and line type.

USAGE

```
declare metering_gate_$comm_chan_star_list entry (char (*), fixed bin,
      pointer, pointer, fixed bin (35));

call metering_gate_$comm_chan_star_list (starname, version, area_ptr,
      chan_star_list_ptr, code);
```

ARGUMENTS

starname
: is the starname to be matched. (Input) Its length must not be more than 32 characters. See the description of starnames in the MPM Reference Guide for information on the construction of starnames.

version
: must be 1. (Input)

area_ptr
: is a pointer to an area in which the channel information is to be allocated. (Input) It must be a nonnull pointer.

chan_star_list_ptr
: is a pointer to the structure in which the channel information is returned. (Output) See "Notes," below, for a description of this structure.

code
: is a standard system status code. It may have one of the following values:

  error_table_$badstar
  : starname is not a valid starname.

  error_table_$nomatch
  : no channel names were found that match starname.

  error_table_$noalloc
  : there was insufficient space to allocate the structure for the channel information.

  error_table_$unimplemented_version
  : the version argument (above) did not contain the current version of the information structure.

  0
  : no errors. (Output)

metering_gate_$comm_chan_star_list

NOTES

The structure pointed to by chan_star_list_ptr has the following format (described in chan_star_info.incl.pl1):

```
dcl 1 chan_star_info based (chan_star_list_ptr) aligned,
    2 version fixed bin,
    2 n_channels fixed bin,
    2 chan_entry (chan_star_count refer (chan_star_info.n_channels)),
      3 name char (32),
      3 mpx_type fixed bin,
      3 parent_type fixed bin,
      3 line_type fixed bin;
```

STRUCTURE ELEMENTS

version
is the version number of the structure.

n_channels
is the number of channels matching the starname.

chan_entry
contains information for each channel matching the starname.

name
is the name of the channel.

mpx_type
is the multiplexer type of the channel. It may have any of the values defined in mulitplexer_types.incl.pl1.

parent_type
is the multiplexer type of the channel's parent multiplexer. If the channel is a level-1 multiplexer, parent_type is -1.

line_type
is the line type of the channel if the channel is a physical subchannel of an FNP, in which case it may have any of the values defined in line_types.incl.pl1; otherwise, it is 0.

phcs_$get_comm_meters

Name: phcs_$get_comm_meters

This entry is used to copy communications metering information for a specified channel from ring 0. Logical channel meters for the specified channel are returned, as are any multiplexer-specific meters maintained for the channel by its own multiplexer module or that of its parent.

USAGE

```
declare phcs_$get_comm_meters entry (char (*), pointer, fixed bin (35));

call phcs_$get_comm_meters (chan_name, info_ptr, code);
```

ARGUMENTS

chan_name
: is the name of the channel. (Input)

info_ptr
: is a pointer to a structure of the same form as that described for the get_meters control order described in the tty_ description in the *Multics Subroutines and I/O Modules* manual, Order No.: AG93. (Input)

code
: is a standard system status code. (Output)

APPENDIX A

DIRECTIONS FOR SETTING UP SYSTEM-SUPPLIED MULTIPLEXERS

This appendix describes various aspects of the administration and use of the system-supplied multiplexers on Multics:

- HASP workstations or HASP hosts
- IBM3270
- polled VIP
- software-simulated terminals
- X.25 network connections
- foreign system connections through a protocol converter

ADMINISTRATION AND USE OF HASP WORKSTATIONS AND HOST SYSTEMS

Multics provides support for the HASP communications protocol. Multics can be configured to operate either as the host or as the workstation. As a host, Multics communicates with a remote workstation that is usually a remote- job-entry (RJE) terminal with an operator's console and one or more card readers, line printers, and card punches. As a workstation, Multics simulates an RJE terminal and communicates with a remote host system.

The HASP communications protocol is supported by using the demultiplexing features of the Multics Communications 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. Each device must be configured as a slave channel.

A process using a device channel should attach that channel through either the hasp_workstation_ or hasp_host_ I/O module described in the *Multics Subroutines and I/O Modules* manual, Order No.: AG93.

The FNP Core Image

Support of the HASP communications protocol requires that the proper software be loaded into the FNP supporting the terminal or remote system. The two modules, bsc_tables and hasp_tables, must be bound into the core image before the FNP is loaded. (See Section 6 ("FNP Core Images") and the description of the bind_fnp command in Section 7.)

Definition of HASP Channels

HASP channels, like all communications channels, are defined in the Channel Master File (CMF). An entry must be made in the CMF for the remote workstation or host (the multiplexer channel) and for each of the individual devices. A sample excerpt from a CMF is shown below.

```
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;
 line_type: BSC;

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

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

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

The above configuration defines a remote workstation with an operator's console, three card readers, five line printers, and a single card punch. For the major channel (a.h014), the line_type, service, and multiplexer_type must be specified exactly as shown. The terminal type is used to distinguish between a connection to a remote workstation and one to a host and to specify options controlling the connection. (This mechanism is described in detail in "Multiplexer Terminal Types," below.)

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 name final component 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 may not 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.)

Multiplexer Terminal Types

All terminal types, including multiplexer terminal types, are defined in the Terminal Type File (TTF). The standard Multics TTF does not contain any HASP multiplexer terminal types.

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).

The configuration information 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 might appear as follows:

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

The complete set of parameters that may be specified is shown below (defaults are provided for all omitted parameters).

name:	type
values:	workstation or host
default:	workstation
meaning:	specifies whether this multiplexer is connected to a remote workstation or a remote host.

name:	block_size
values:	400 through 1017
default:	400
meaning:	specifies the maximum block size, in characters, used by Multics for all messages transmitted on the multiplexer channel.

name: signon_mode
values: 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 may be transmitted; this parameter is ignored when connecting to a workstation.

name: multileave_mode
values: yes or no
default: yes
meaning: specifies that blocks transmitted by Multics may contain records from more than one device. This parameter may need to be turned off for communications with some workstations.

name: suspend_all_mode
values: 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 turned on for communications with some workstation (e.g., an IBM System/370 running the HASP workstation simulation option of RSCS under VM/370).

name: rts_mode
values: yes or no
default: yes if type is host; no if type is workstation
meaning: specifies whether Multics will request permission from the remote host or workstation before transmitting data on other than the operator's console.

name: connect_timeout
values: 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 will either wait or send the initial connection sequence indefinitely as appropriate.

name: receive_timeout
values: 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.

name: transmit_timeout
values: 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.

name: max_naks
values: 5 through 100
default: 10
meaning: specifies the maximum number of consecutive NAKs that may be transmitted or received by the FNP on the communications channel without an intervening block before aborting the connection.

name: max_device_input_records
value: 3 through 30
default: 6
meaning: specifies the maximum number of records that the multiplexer will hold 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.

Subchannel Terminal Types

All data transmitted over a HASP subchannel must be both 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 the *Multics Subroutines and I/O Modules* manual, Order No.: AG93. A terminal type may be specified in the attach descriptions of these I/O modules to define the remote host's or workstation's character set. See the section "Character Set Specification" in the descriptions of these I/O modules for more information.

Control Orders Used by HASP Subchannels

The following control orders are recognized for HASP subchannels in addition to the control orders defined in the description of the tty_ I/O in the *Multics Subroutines and I/O Modules* manual, Order No.: AG93. These control orders are intended to be used exclusively by the hasp_workstation_ and hasp_host_ I/O modules and should not be invoked by user programs.

In the following descriptions, the "info_ptr" field supplies the declaration of the item that must be supplied by the caller to the HASP multiplexer software. An indication is also supplied as to whether the caller must also supply any information in the field before issuing the control order (Input vs. Output).

name: get_device_type
function: 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)
notes: the possible values returned by this control order are defined in the include file hasp_device_data_incl.pl1.
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.

name: signon_record
name: no_signon_record
function: see the section "SIGNON Processing" in the write-up of the hasp_host_I/O in the *Multics Subroutines and I/O Modules* manual, Order No.: AG93.

ADMINISTRATION AND USE OF IBM3270 TERMINALS

Multics provides support for IBM Model 3271 terminal systems as login devices. These systems comprise a cluster of keyboard/display stations and receive-only printers that can be connected to Multics via a single shared communications line. Multics uses the IBM3270 bisync protocol to control these devices.

NOTE: When logging in to Multics on a 3270 device, Multics does not prevent the password from being displayed, either by masking it or turning off the printer.

The IBM3270 terminal system is supported using the demultiplexing features of the Multics Communications System. This allows each device on the controller to be controlled independently by separate processes. Each keyboard/display station can be configured as either a login device or a slave channel; receive-only printers can only be configured as slave channels.

Note that configuring an IBM3270 channel for use as a multiplexer is incompatible with the existing ibm3270_ I/O module, which requires that the channel be configured as a slave. A 3270 terminal system can be controlled by a single process as a slave device by using ibm3270_, or can be controlled by multiple processes as login devices using the IBM3270 demultiplexer.

The FNP Core Image

Support of IBM3270 terminals requires that the proper software be loaded into the FNP supporting the terminal. The two modules required are bsc_tables and ibm3270_tables. Both of these modules must be bound into the core image before the FNP is loaded. (See Section 6 ("FNP Core Images") and the description of the bind_fnp command in Section 7.)

Definition of IBM3270 Channels

IBM3270 channels, like all communications channels, are defined in Channel Master File (CMF). An entry must be made in the CMF for the subsystem or controller and for each of the subchannels. A sample excerpt from a CMF is shown below.

```
name: a.h012;        /* entry for controller */
 baud: 9600;
 line_type: BSC;
 service: multiplexer;
 multiplexer_type: ibm3270;
 terminal_type: IBM3271;

name: a.h012.d00-a.h012.d03;  /* entry for 4 displays */
 attributes: hardwired, dont_read_answerback;
 line_type: BSC;
 service: login;
 terminal_type: IBM3277_M2;

name: a.h012.p04;    /* entry for 1 receive-only printer */
 line_type: BSC;
 service: slave;
 terminal_type: IBM3284_M2;
```

The above configuration defines a system with four keyboard/display stations and one printer. For the major channel (a.h012), the line_type, service, and multiplexer_type must be specified exactly as shown. The terminal type is used to distinguish between various types of 3270 subsystems, and is explained below.

The names of the subchannels (a.h012.*) must be in a specific format. The final component must be exactly three characters. The first character must be "p" for printer channels or "d" for keyboard/display stations. The final two characters must be two decimal digits that define the device address. The multiplexer cannot be configured with a printer and a display that have the same address.

Multiplexer Terminal Types

The terminal type for the multiplexer channel specifies options to distinguish between various kinds of 3270 systems. The following standard multiplexer terminal types are defined in the Terminal Type File (TTF) (additional types may be defined by an administrator, as required):

```
IBM3271
IBM3271_ASCII
```

The IBM3271 terminal type is for use in configurations using the EBCDIC bisync protocol. The IBM3271_ASCII type is for use with the ASCII protocol.

The only portion of the TTF entry for the multiplexer terminal type that is used is the "additional_info" keyword. The character string is used to specify the value of various parameters used to control multiplexer operation. The field is formatted as a series of parameter assignments, separated by spaces. Each parameter assignment has the following form:

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

A sample TTF entry for a multiplexer terminal type might appear as follows:

```
terminal_type: IBM3271;
 additional_info: "controller_address=1 quit=pa2";
```

The complete set of parameters that may be specified is shown below (defaults are provided for all omitted parameters).

name:	controller_address
values:	0 through 31
default:	0
meaning:	specifies the device address of the controller to be used for polling. (Multiple controller configurations on a single communications channel are not supported.)

name:	quit
values:	pa1, pa2, or pa3
default:	pa1
meaning:	specifies which function key is to be interpreted as the Multics quit function.

name:	formfeed
values:	pa1, pa2, pa3, or clear
default:	clear
meaning:	specifies which function key is to be interpreted as restarting output on an end-of-page situation.

name:	code
values:	ascii or ebcdic
default:	ebcdic
meaning:	specifies which character code is to be used.

name: allow_raw3270
values: yes or no
default: yes
meaning: specifies whether users of subchannels of this multiplexer may use the special mode "raw3270". This mode is intended for forms control and is explained in more detail below.

name: allow_copy
values: yes or no
default: no
meaning: specifies whether or not users of raw3270 mode may make use of the 3270 copy function. Enabling this function allows a user of one subchannel to "copy" the memory of another subchannel. This could, in malicious hands, compromise the passwords of users on other subchannels.

name: debug
values: yes or no
default: no
meaning: specifies that the ring-0 portion of the 3270 multiplexer may write certain debugging information on the syserr console. This is used only by demultiplexer developers.

Subchannel Terminal Types

Terminal types for subchannels are also provided in the standard TTF. The following terminal types, which correspond to IBM model numbers, are provided:

```
displays: IBM3277_M1
          IBM3277_M2
          IBM3277_M1_ASCII
          IBM3277_M2_ASCII
printers: IBM3284_M1
          IBM3284_M2
```

Typing Conventions

A 3270 subchannel is not considered usable (or dialed) until some input is received from it. Therefore, the first step in using a 3270 keyboard/display is to activate or press any of its input keys (usually "enter", but "clear", or any of the "pa" or "pf" keys may be used also). In response to this first input (which is discarded), Multics clears the screen and sends the standard greeting message.

The 3270 demultiplexer uses an unformatted screen in controlling the display. It also controls the keyboard unlocking feature and only unlocks the keyboard when the process is ready for more input. Unlocking of the keyboard can be regarded as a prompting function; the previous input has been acted upon, and the process is ready for input again.

When typing input on a 3270 display unit, the following rules must be followed:

1. Always begin typing where the system positions the cursor.

2. Always leave the cursor after the last character of input. Local editing functions of the terminal may be used to edit an input line, but the cursor must always be returned to the end of the line of input.

3. Always terminate input with the "enter" key.

4. Never attempt to send more than one line of input using the return key of the terminal. Single lines of input longer than the line length of the terminal may be sent normally by allowing the terminal to wrap at the end of the line (except on the last line of the screen). Multiple lines are not possible.

5. When typing over a password mask (printer_off is not supported), either space out over the end of the mask (after the password is typed), or use the erase_eof function to delete the end of the mask. One of these options must be performed before using the "enter" key. Failure to do this properly results in part of the password mask being interpreted as input.

6. The terminal's tab key may not be used to enter tabs into the input text. The only way to enter a tab is to use the escape sequence ¢11.

These typing conventions must be followed exactly. Failure to do so may cause input to be ignored, lost, or misinterpreted.

The formfeed function code (normally pa1) as defined in the multiplexer terminal type has two uses. First, when output is suspended at the end of the screen (EOP on the last line), the formfeed causes the next page of output to be sent. If the terminal is not in an EOP situation, it causes the screen to be cleared and new input and output to begin at the top of the screen.

The 3270 does not implement the full ASCII character set. Therefore, the following conventions are used:

ASCII	3270
\	¢ (cent sign)
[	¢<
]	¢>
{	¢(
}	¢)
`	¢'
~	¢t

These conventions are the same as used by Multics for IBM2741 terminals.

raw3270 Mode

The 3270 demultiplexer implements a special mode named "raw3270". This mode is set and reset like all other tty modes, using the same command and subroutine interfaces. This mode is intended for forms programs, and is used to make the full set of 3270 features available to the user. The mode may be set by the user (if allowed by the multiplexer terminal type, see above) at any time. It does not go into effect, however, until rawo and rawi modes are also set. When raw3270 mode is in effect, the user is expected to format a complete 3270 bisync message, including the STX and ETX, but excluding the leading SYN and the trailing block check characters. This message, with some minimal checking, is sent unchanged to the terminal. The function code (following the STX-ESC) may be either write, erase_write, or copy (if allowed by the multiplexer terminal type). Read_buffer, read_modified, and erase_all_unprotected are not supported.

When raw3270 mode is in effect, input is not interpreted by the multiplexing software (except for messages caused by the TEST REQ key); it is passed to the user exactly as received from the terminal. The only change that occurs is to combine multiple input messages from the terminal into a single bisync message when the terminal has split its input into multiple records because it exceeded 256 characters.

Also, when raw3270 mode is in effect, the hndlquit mode takes on an additional meaning. If hndlquit is on, the quit key defined for the multiplexer is processed normally; this function key is unavailable for user programs. If the channel is in ^handlquit mode, no quit processing takes place and the quit key may be used by the application as it sees fit. Note that in ^hndlquit,raw3270 mode there is no way to perform a Multics quit function.

Special care must be taken by the application program when operating in raw3270 mode. Any unexpected output to the terminal (e.g., messages from other users, memos, and error messages) will be ill-formatted, may not appear on the user terminal, and may have unpredictable side effects. It is important for the application program to anticipate all possible error conditions, because the default error handler may not return the terminal to a usable state when a condition occurs. Thus, in raw3270 mode, when an error occurs, and the user is returned to command level, there may be no way for him to enter a command since the process is expecting complete bisync messages from the terminal.

ADMINISTRATION AND USE OF POLLED VIP TERMINALS

The Multics Communications System provides support for several Honeywell VIP terminal subsystems including the VIP7700, VIP7760, and VIP7804. These subsystems typically comprise a number of keyboard/display stations and receive-only printer stations that can be connected to Multics via a single communications line. The Honeywell polled VIP communications protocol supports these terminals.

The Multics Communication System contains a multiplexing mechanism that allows each station of a polled VIP subsystem to be independently controlled. Thus, each keyboard/display station can be used as a separate login terminal. Similarly, each printer station can be used as a slave device attachable by any authorized process, such as an I/O daemon (see Multics Bulk I/O for information about I/O daemons).

The FNP Core Image

Support of polled VIP terminals on Multics requires that a particular software module, known as polled_vip_tables, be present in the FNP core image. To accomplish this, the bindfile must be changed to include polled_vip_tables. A new core image must then be generated to replace the previous one. For further details, see Section 6 ("FNP Core Images") and the description of the bind_fnp command in Section 7.

Definition of Polled VIP Channels

A channel must be defined in the Channel Master File (CMF) for each terminal subsystem, i.e., for each polled VIP communications line connected to Multics. This channel is known as a multiplexer channel because it connects the entire subsystem. For each station in the subsystem, another channel must be defined in the CMF. These channels are referred to as subchannels of the multiplexer channel. A sample excerpt from the CMF is shown below in which a multiplexer channel and various subchannels are defined. For a complete description of the CMF, see Section 4.

```
name: a.h005;
 baud: 2400;
 dataset: 201C;
 line_type: POLLED_VIP;
 service: multiplexer;
 multiplexer_type: vip7760;
 terminal_type: VIP7700_CLUSTER;

name: a.h005.d00-a.h005.d01;
 attributes: dont_read_answerback;
 service: login;
 terminal_type: VIP7705;

name: a.h005.p01;
 service: slave;
 baud: 1200;
 terminal_type:  VIP7714;
```

The above example shows a polled VIP multiplexer with three subchannels: two displays and one printer. It is essential that the line_type and multiplexer_type be specified as shown. The multiplexer_type of vip7760 is used generically for all polled VIP multiplexer channels. It does not imply that the subsystem must be a VIP7760. The terminal_type of the multiplexer channel distinguishes the different types of subsystems. More will be said about this later.

The names of subchannels are chosen according to specific rules. The final component of the subchannel name must be exactly three characters. The first character must be "d" for a keyboard/display station, "p" for a printer station, or "x" for a transparent communications channel using the polled VIP protocol. The last two characters represent the station poll address. Note that a keyboard/display station can share the same poll address with a printer station.

A baud rate should be specified for printer subchannels to distinguish between 300 and 1200 baud printers.

Printer subchannels should not be defined for a VIP7804 subsystem. On the VIP7804, a printer is treated as a peripheral device of the keyboard/display station to which it is connected. The printer can only be addressed by the use of escape sequences within the text sent to the keyboard/display station. Therefore, VIP7804 printers cannot be supported as independent subchannels.

Multiplexer Terminal Types

All terminal types, including multiplexer terminal types, are defined in the Terminal Type File (TTF). The standard Multics TTF contains three polled VIP multiplexer terminal types that correspond to three different subsystems:

```
VIP7700_CLUSTER
VIP7760_CONTROLLER
VIP7804_CLUSTER
```

The differences among polled VIP subsystems have been categorized according to a set of parameters that control the behavior of the polled VIP multiplexer software within the Multics Communication System. The additional_info field of the multiplexer terminal type is used to specify the values of these parameters. This field is formatted as a character string containing a sequence of parameter assignments separated by spaces. Each parameter assignment has the following form:

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

A sample multiplexer terminal type would appear in the TTF as follows:

```
terminal_type: VIP7700_CLUSTER;
additional_info: "controller_poll=no quit=q formfeed=1";
```

Default values apply to any parameters not specified or to all parameters if no terminal type is specified. The complete set of parameters is described below.

name:	controller_poll
values:	yes, no
default:	no
meaning:	if yes, indicates that the subsystem responds to a controller poll.

name: crlf_echo
values: yes, no
default: no
meaning: if yes, indicates that Multics echoes a carriage return/linefeed sequence after each input message is received.

name: omit_nl
values: yes, no
default: no
meaning: if no, Multics appends a newline character to the end of each input message.

name: omit_ff
values: yes, no
default: no
meaning: if no, Multics sends a formfeed character to clear the screen before resuming output after an end-of-page condition.

name: gcos_break
values: yes, no
default: no
meaning: if yes, Multics recognizes an input message containing the string $*$BRK as a quit indication.

name: etb_mode
values: yes, no
default: no
meaning: if yes, Multics terminates partial message blocks transmitted to the terminal with an ETB character instead of an ETX character. This ensures that escape sequences that are split across blocks are correctly interpreted by the terminal. (Note: The VIP7700 does not support etb_mode.)

name: pause_time
values: integer N where N >= 0
default: 1000
meaning: specifies the pause time in milliseconds between consecutive poll cycles.

name: max_text_len
values: integer N where 508 <= N <= 1920
default: 1024
meaning: specifies the maximum number of data characters in any message transmitted by Multics to the terminal.

name: max_message_len
values: integer N where 73 <= N <= 1024
default: 289
meaning: specifies the maximum number of characters expected in an input message from the terminal. This parameter is normally not specified except on channels used for Level-6 file transfer.

name: quit
values: any single ASCII character
default: q
meaning: when this character appears in the first function code position of an input message, it indicates a quit.

name: formfeed
values: any single ASCII character
default: l
meaning: when this character appears in the first function code position of an input message, it indicates a formfeed operation.

Subchannel Terminal Types

Terminal types for polled VIP subchannels are specified in exactly the same way as terminal types for ordinary asynchronous channels. The standard TTF contains three terminal types that apply to polled VIP keyboard/display subchannels:

```
VIP7705   /* VIP7700 display with upper/lower case */
VIP7760   /* VIP7760 display */
VIP7804   /* VIP7804 display */
```

Printers for polled VIP subsystems are one of two basic types: TermiNet or Rosy. The standard TTF contains two corresponding terminal types named TN300 and ROSY. Also, a VIP7714 terminal type is defined that is essentially equivalent to TN300.

For keyboard/display stations used as login terminals, the CMF subchannel entry should specify the appropriate terminal type. For slave subchannels (normally printers), the terminal type must be set after the device is attached.

Input Size Considerations

A single transmission from a polled VIP terminal can contain any number of characters and any number of lines. In the normal case, Multics expects to receive exactly one line with each transmission. When page-length mode is enabled, Multics counts each separate transmission as one line of input. Therefore, if more than one line is actually sent, Multics miscalculates the true cursor position. This can cause subsequent output to be improperly formatted. Therefore, normal use of a VIP7700 or VIP7760 terminal requires pressing the transmit key at the end of each line of input; the return key is not used. On the VIP7804, a mode of operation called "transmit on return" is provided that causes each line to be transmitted when the return key is pressed. This mode is recommended and its use is assumed in the discussions that follow.

If page-length mode is disabled for some special purpose, e.g., a forms processing application, the single-line input restriction can be ignored. In this case, there is no problem in transmitting multiple lines or an entire screen.

Polled VIP terminals (or subsystems) have a strapping option that limits the maximum transmission block size to 256 data characters. Larger transmission requests are divided into a sequence of blocks. Multics expects this option to be selected and rejects input blocks containing more than 256 data characters, unless the max_message_len parameter (above) is specified.

Function Codes

Function codes are ordinary characters that occupy a special position within the polled VIP message structure. Each time a polled VIP terminal transmits a message to Multics, the message contains two function codes. On the VIP7700 and VIP7760, these function codes can be set from the keyboard by pressing the control (CTL) key in combination with a regular key. On the VIP7804, function codes are generated only by the 12 special-function keys at the top of the keyboard.

Multics uses the first function code in a message as a means for indicating a quit or formfeed operation. This is necessary because the normal mechanisms for indicating these operations are not available. Like most other synchronous terminals, a polled VIP terminal does not transmit a line break and, consequently, does not have a break key. Also, a polled VIP terminal does not transmit certain ASCII control characters including formfeed.

The transmission of function codes varies from one terminal to another. On the VIP7700 and the VIP7760, function codes, like data, are not transmitted until the transmit key is pressed. The VIP7700, however, does not transmit a function code unless data has been entered. Therefore, a VIP7700 user must type a data character in order to send a function code. Multics ignores any data that arrives in the same message with a recognized function code. The VIP7804 is very different: when a special function key is pressed, a message containing a function code is transmitted automatically without any need to press the transmit key.

QUITS

As mentioned above, a polled VIP terminal does not have a break key. Therefore, a function code is used to indicate a quit. On the VIP7700 and VIP7760, the quit function code is normally "q". This code is generated by simultaneously pressing the control key and the Q key. On the VIP7804, the quit function code is normally an underscore (_). This code is generated by simultaneously pressing the shift key and the F12 function key. The quit function code can be changed by use of the quit parameter described earlier.

Since a quit discards output that has already been formatted by Multics, the cursor position computed by Multics is made invalid. This can cause subsequent output to be improperly formatted. Therefore, whenever a user issues a quit while output is pending, the user should next issue a formfeed to resynchronize Multics with the true cursor position. Normally, this is only necessary if the user quits while at end of page.

FORMFEEDS

As mentioned above, a polled VIP terminal does not transmit a formfeed character. Therefore, a function code is used to indicate a formfeed operation. On the VIP7700 and VIP7760, the formfeed function code is normally "l". This code is generated by simultaneously pressing the control key and the L key. On the VIP7804, the formfeed function code is normally a circumflex (^). This code is generated by the F12 function key (unshifted). The formfeed function code can be changed by use of the formfeed parameter described earlier.

The formfeed function code is used to request Multics to clear the screen. This is often done before executing a command that produces a full page of output to avoid dividing the output between two pages. Although the user can clear the screen locally by use of the clear key, Multics is unaware of the change in cursor position. This causes subsequent output to be improperly formatted. Therefore, it is recommended that the clear key not be used.

End Of Page

Normal use of a polled VIP terminal requires that page- length mode be enabled. When an end-of-page condition is encountered, the resumption of output can be requested in several different ways. One way is to send a formfeed function code. Unfortunately, function codes are not especially convenient to send on the VIP7700 or VIP7760 (several keystrokes are required). Since the end-of-page condition occurs often in normal use, a more convenient method of resuming output is provided.

When at end of page, the transmission of any message to Multics is construed as a request to resume output. Therefore, on the VIP7760, a user need only press the transmit key. This sends an empty message to Multics. The VIP7700, however, does not transmit an empty message. Therefore, at least one data character must be typed. The commercial-at sign (@) is used for this purpose. When at end of page, if Multics receives a message containing only the "@" character, output is resumed and the message is discarded. If the message has any other contents, it is treated as normal input, and output resumes.

On the VIP7804, one can use either the return key or the formfeed function key to resume output.

Blank Lines

Typing a blank line requires merely pressing the transmit key on the VIP7760 or the return key on the VIP7804. The VIP7700, however, does not transmit a blank line; on this terminal it is necessary to type a space before pressing the transmit key.

Tabs

Use of the tab key on a polled VIP terminal produces different effects from an asynchronous terminal. As with most buffered terminals, a tab is converted to the appropriate number of spaces when displayed on the screen. These spaces are what eventually get transmitted to Multics. Hence, a polled VIP terminal never transmits a tab control character.

An exception arises when a user types a tab at the beginning of a line. In this case, the tab is interpreted not as a shorthand for spaces, but rather as a shorthand for moving the cursor right. The difference is that no space characters are entered on the screen, and, therefore, no spaces are transmitted. If the user really wants spaces, the problem can be avoided by always typing a space followed by a tab when at the beginning of a line.

In order to create and edit text containing tabs, it is necessary for a polled VIP user to type an escape sequence that Multics translates into a tab character. Normally, the backslash h (\h) sequence is used to represent a horizontal tab. This sequence is defined by the special-characters table associated with a given terminal type.

Circumflex and Tilde

On the VIP7700 and VIP7760, the circumflex (^) and tilde (~) characters have special interpretations. The circumflex character causes blinking of subsequent characters on the same line, up to the first space. The tilde character causes blanking of subsequent characters on the same line, up to the first space. Multics, however, attaches no such special interpretation to these characters. Therefore, when Multics transmits these characters to the terminal, it is necessary to represent them without producing the blinking or blanking effect.

On the VIP7760, this is accomplished by immediately following each circumflex or tilde with a space. This same technique cannot be used on the VIP7700 because it does not display the circumflex or tilde. Therefore, on the VIP7700, a circumflex is represented by the backslash minus sign (\-) sequence and tilde is represented by the backslash equal sign (\=) sequence. These sequences can also be typed as input on both the VIP7700 and VIP7760.

Dialups and Hangups

Individual stations on a polled VIP Subsystem do not "dial up" or "hang up" a communications line as do ordinary terminals. Therefore, these actions are simulated to some extent. At the time a subsystem is initialized, all slave subchannels are assumed to be dialed up, and all login subchannels are assumed to be hung up. To dial up a keyboard/display station, a user must send an input message of any kind. This message is discarded by Multics, but causes a simulated dialup to occur. Multics responds by sending a greeting message to the station. The user can then log in as usual.

There is no way to produce a hangup condition from a polled VIP station. Switching off power to the station has no effect. Therefore, the only way to terminate a Multics session is by use of the logout command. After logout, a station reverts to the hung-up state. To begin a new session, the dialup and login procedure is repeated as described above.

ADMINISTRATION AND USE OF SOFTWARE-SIMULATED TERMINALS

Multics provides support for software-simulated terminals. These are "pipes" with simulated Multics terminal interfaces at either end. In the usual case, one end is configured as an autocall channel and the other as either a login or slave channel.

Definition of Software Terminal Channels

Software terminals, like all other communications channels, are defined in the Channel Master File (CMF). An entry must be made in the CMF for the software terminal facility, itself, as well as two for each pipe to be defined. A sample CMF excerpt is shown below.

```
name: sty;                                  /* entry for facility */
 service: multiplexer;
 multiplexer_type: sty;
name: sty.slogin_001-sty.slogin_020;        /* entry for server side
                                            of login channels */
 service: login;
 terminal_type; STY;

name: sty.sslave_001-sty.sslave_005;        /* entries for server
                                            slave channels */
 service: slave;
 terminal_type: STY;

name: sty.ulogin_001-sty.ulogin_020;        /* entries for user side
                                            of login channels */
 service: autocall;
 terminal_type: STY_USER;

name: sty.uslave_001-sty.uslave_005;        /* entries for user sides
                                            of slave channels */
 service: autocall;
 terminal_type: STY_USER;
```

The above configuration defines a system with 20 software terminals for login service and five for slave service. Each one has two channels associated with it, the "server" and "user" sides. The server and user sides must have the same name, except for the first character of the last component, which should be "s" for servers and "u" for users.

If you type ahead on a login service channel that expects an answerback, the typed-ahead data is interpreted as the answerback string, causing STY to be hung up. You can avoid this circumstance by specifying the dont_read_answerback attribute for the login STY.

Multiplexer Terminal Types

The terminal types associated with software terminals (STY and STY_USER) define the conversions appropriate to these channels.

Use of software-simulated terminals is incompatible with the access isolation mechanism (AIM).

SPECIFICATIONS FOR AND ADMINISTRATION OF X.25 NETWORK CONNECTIONS

The Multics Communications System provides support for a connection to a packet switching network through an interface complying with 1980 CCITT Recommendation X.25. The networks certified for use include TYMNET, DATAPAC, TELENET, and UNINET.

The Multics Communications System contains a multiplexing mechanism that allows each logical channel through the network to be independently controlled. In this manner, it is possible to configure some logical channels as login terminals, while others may be used for outgoing access.

Hardware Requirements

The X.25 interface runs only on a DCU6661 FNP with at least 64K words of memory. Each separate X.25 connection requires an HDLC interface feature. Those supported are:

- DCF6620 0-9600 bits/sec V.24 interface (RS232-C)
- DCF6623 0-72000 bits/sec V.35 interface
- DCF6622 0-72000 bits/sec Bell 3XX interface

Software Support

The X.25 communications protocol is an international standard that defines an interface to various public data networks. The information below describes various aspects of the X.25 interface as implemented on Multics systems. It is assumed that the user is familiar with the terminology used in defining the X.25 international standard.

LINK LEVEL (X.25 LEVEL 2)

Both the LAPB and LAP link-level procedures are supported. Multics can act as either a DTE or DCE. The link-level parameters can be controlled on a per-link basis as follows:

```
Timer (T1)                                  0-51.1 sec in units of .1 sec.
Timer (T3)                                  1-511 sec
Maximum Number of Transmissions (N2)        1-511
Maximum Number of Bits in an I Frame (N1)   24-8232 in multiples of 8 bits
Maximum Number of Outstanding I Frames (k)  1-7
```

The link-level parameters are specified in the additional_info keyword used with the TTF entry for X.25 channels. See "Terminal Type File" below.

PACKET LEVEL (X.25 LEVEL 3)

Permanent Virtual Circuit and Datagram service are not supported. Up to 4095 virtual circuits can exist. A "virtual circuit" is the equivalent of a logical channel. The maximum User Data field length can be any of 64, 128, 256, 512, and 1024 octets and must be the same for all virtual circuits. The window size can be chosen from 1-7 and must be the same for all virtual circuits. User Data field length and window size are defined in the

additional_info keyword used with the TTF entry for X.25 channels. See "Terminal Type File" below. Use of the More Data Mark (M bit) is not supported. Use of the Qualifier (Q) bit other than by X.29 is not supported. The following Optional User Facilities are not supported:

- Extended Packet Sequence Numbering
- Packet Retransmission
- Closed User Group
- RPOA Selection
- Flow Control Parameter Negotiation
- Throughput Class Negotiation

TERMINAL CONTROL LEVEL

In general, the terminal control protocol described by the 1980 CCITT recommendations X.3 and X.29 is supported. The X.3 parameters supported and corresponding Multics terminal modes are described below. (Multics terminal modes are specified as part of the TTF entry used to define each terminal. See the *Multics Programmer's Reference Manual*, Order No.: AG91, for a description of the TTF.)

1	PAD recall by escaping from the data transfer state	rawi
2	Echo	echoplex
3	Selection of data forwarding signal	always set to 126
4	Selection of idle timer delay	breakall
5	Ancillary device control	iflow
7	Selection of operation of PAD on receipt of break signal from the start-stop mode DTE	hndlquit
8	Discard output	used by interrupt procedure
12	Flow control of the PAD by the start-stop mode DTE	oflow
13	LF insertion	lfecho

In addition, the following DATAPAC national parameters are supported:

125	Output pending timer	polite

Implementing an X.25 Capability on Multics

In order to obtain an X.25 capability on Multics, the user must:

1. Add appropriate X.25 software to the FNP.
2. Define the X.25 connection (physical channel) and the various terminals to be associated with the channel.
3. Install a specially defined Terminal Type File (TTF).

A description of each procedure is listed below.

THE FNP CORE IMAGE

Support of an X.25 connection on Multics requires that the software module "x25_tables" be present in the FNP core image. To accomplish this, the bindfile must be changed to include x25_tables, and a new core image must be generated. For further details see Section 6 ("FNP Core Images") and the description of the bind_fnp command in Section 7.

DEFINITION OF X.25 CHANNELS

One channel must be defined in the Channel Master File (CMF) for each X.25 network connection. Appropriate subchannels must be defined for each terminal to be associated with the multiplexed channel. A sample CMF follows.

```
name: b.h102;
 baud: 9600;
 line_type: X25LAP;
 service: multiplexer;
 multiplexer_type: x25;
 terminal_type: X25_DATAPAC;

name: b.h102.001-b.h102.050;
 service: login;
 terminal_type: ASCII;

name: b.h102.051-b.h102.063;
 service: autocall;
 terminal_type: none;
```

The above configuration defines a connection to the network at 9600 baud through channel b.h102. Multics will behave as the DTE (the default type) and will use the basic X.25 protocol. There are 50 inward (login) channels and 13 outward channels.

TERMINAL TYPE FILE (TTF)

The user must install a specially defined TTF to support the X.25 facility. The TTF must contain a site-specific entry for the X.25 connection. Examples of entries for TYMNET and DATAPAC are supplied in the TTF shipped with the system.

The only portion of the TTF entry that is used is the "additional_info" keyword. The character string is used to specify the value of various parameters used to control multiplexer operation. The parameters are all optional except "n_lc," which must be specified. The field is formatted as a series of parameter assignments, separated by spaces.

A sample TTF entry for a X.25 multiplexer might appear as follows:

```
terminal_type:    X25_TYMNET
additional_info: "network=tymnet n_lc=32 packet_size=128 window_size=2
                  type=DTE link_protocol=LAP frame_size=8232 T1=3 N2=20
                  T3=3 K=7";
```

The complete set of parameters is listed below:

name: type
values: DTE, DCE
default: DTE
meaning: specifies whether the Multics system behaves as the DTE or DCE.

name: link_protocol
values: LAP, LAPB
default: LAP
meaning: connection mode is either LAP or LAPB (Balanced).

name: disc_first
values: yes, no
default: no
meaning: specifies whether Multics enters the DISC SEND state. This should only be used for networks implementing the old TELENET X.25 protocols.

name: frame_size
values: 24 through 8232 in multiples of 8
default: 1064
meaning: specifies the maximum size, in bits, of transmitted frames. This size must be large enough to contain a packet whose length is packet_size (see below) plus the level 3 header (3 or 4 characters).

name: K
values: 1 through 7
default: 7
meaning: specifies the number of outstanding unacknowledged frames allowed.

name: N2
values: 1 through 511
default: 20
meaning: specifies the number of times a frame is retransmitted before the link is reset.

name: T1
values: .1 through 51. (in tenths of a second:)
default: 3.0
meaning: specifies the time, in seconds, to wait for a frame acknowledgement.

name: T3
values: 1 through 511
default: 3.0
meaning: specifies the number of seconds to wait for a response to a link reset.

name: packet_size
values: 64 through 1024
default: 128
meaning: specifies the maximum number of octets (characters) to be transmitted in a single packet.

name: collect
values: yes, no
default: no
meaning: specifies that all outgoing calls from a given x.25 multiplexer are to be made collect; i.e., the receiving system is to pay the network charges.

name: packet_threshold
values: 2 through 1025
default: packet_size+1
meaning: specifies the minimum length of a "long packet". The multiplexer will not output a long packet if there are any short packets queued for output. The effect of this is to optimize the transmission of short packets such as might be used to echo character-at-a-time input. The default value of one greater than the maximum packet size has the effect of disabling the preferential treatment of short packets.

For example, if a number of Emacs users are sharing a network connection with an output-intensive application such as Inter- Multics File Transfer, the site may want to set the packet threshold to a small value in order to give the 1- to 3-character packets used to echo input priority over the large packets generated by the file transfer application.

name: window_size
values: 1 through 7
default: 2
meaning: specifies the window size W to be used in communicating with the network.

name: local_address
values: 1 to 15 decimal digits
default: none
meaning: specifies the local address to be used in call request packets. This parameter has no default value; if no value is specified, none is used.

name: network
values: tymnet, datapac
default: none
meaning: specifies which set of national parameters will be assumed. This parameter has no default value; if no value is specified, none is used.

name: n_lc
values: 1 through 4095
default: none
meaning: specifies the number of X.25 logical channels to be defined. This parameter has no default value and must be specified.

name: d_bit
values: yes, no
default: yes
meaning: specifies whether the network supports the use of the D bit for end-to-end acknowledgement.

name: breakall_idle_timer
values: 1 through 255
default: 2
meaning: specifies the value that will be used to set PAD parameter 4 (the idle timer) when a user enters breakall mode. The value is specified in twentieths of a second. Changing this value will cause users to see different response characteristics in applications which use breakall mode (e.g. video, emacs). The PAD will not forward input until the user has not typed anything for the specified time. This can greatly decrease the number of packets sent for input, and thus the load on the network, the Multics frontend, and Multics itself. The disadvantage of a longer timer is that the user will not see his input echoed until the PAD sends a packet to Multics.

CONNECTING TO A FOREIGN SYSTEM THROUGH A PROTOCOL CONVERTER

A user logged into Multics on an asynchronous terminal can talk to a foreign system by attaching to a protocol converter through subchannels on an FNP. Any incompatibilities between the terminal type and the foreign system are made transparent by the protocol converter. For example, the Renex Protocol Converter, Model RT9B, manufactured by the Renex Corporation of Springfield, Virginia, can be used to connect Multics to an IBM system, by simulating a 3270 terminal controller.

The connection is made through the services of the dial_out facility, by specifying either the dial_out or connect command to a generic destination, thereby causing a privileged_attach of a slave channel with a matching destination. Alternatively, a dial out over an autocall channel can be effected, but this requires that a specific destination be supplied. The dial_out and connect commands are described in detail in the *Multics Commands and Active Functions* manual, Order No.: AG92. The generic_destination statement is part of the CMF channel entry (see below).

Once the connection is established, data transmission may commence between the two nodes at a speed appropriate to the configuration (i.e., the FNP and the protocol converter in use). A wait request is available for use with dial_out exec_coms, which allows for synchronization of transmissions from the foreign system (see the description of the dial_out command in the *Multics Commands and Active Functions* manual, Order No.: AG92).

Channel Definition for Foreign System Connections

Typically, a number of channels are hardwired between the Multics FNP and the protocol converter. Their CMF entries include a generic_destination statement to be specified in the dial_out command. A sample excerpt from the CMF is shown below; note that the channel names can be arbitrary. For a complete description of the CMF, see Section 4.

```
name: a.h006;
terminal_type: ascii;
baud: 9600;
attributes: hardwired, dont_read_answerback;
generic_destination: IBM1;
service: slave;
comment: "connection to IBM system through Renex protocol converter";

name: b.h102;
terminal_type: ascii;
baud: 9600;
attributes: hardwired, dont_read_answerback;
generic_destination: IBM1;
service: slave;
comment: "connection to IBM system through Renex protocol converter";
```

This excerpt defines two channels connected to an IBM system through a Renex protocol converter that simulates a 3270 terminal controller. By entering the dial_out command to the generic destination (dial_out IBM1), the user can attach whichever line is available to communicate with the IBM system.

Mapping the Terminal Type to the Foreign System

Sometimes the channel requires special initialization before communication can be established (this usually depends on the compatibility between the terminal type and the foreign system). Thus, a protocol converter commonly determines the user's terminal type so that any necessary mapping of terminal functions can be performed before the connection is established.

When terminal conditioning is required, the user enters the connect command, rather than the dial_out command, to the generic destination (connect IBM1, for the excerpt above), which automatically executes a site-supplied dial_out exec_com (IBM1.dial_out, in this case) to perform the appropriate startup functions. The connect command bypasses any existing user dial_out start_up exec_com, searching the dial_out search list as follows:

```
-working_dir
>udd>[user project]>dial_out_dir
>site>dial_out_dir
```

A sample dial_out exec_com to start up communications on a line attached through a Renex protocol converter is shown below.

```
&version 2
&trace off
&-
&- Translate terminal type to Renex terminal type number.
&- If the terminal type is not supported by Renex,
&- generate XX for the terminal number.
&-
&set renex_type
   &+ &[e before
       &+ &[e after
           &+ :VIP7200:03:VIP7201:03:VIP7205:03:VIP7801:03
               &+:VIP7803:03:VIP7804:03:ADM3A:06:HAZELTINE1500:02
               &+:HP2621:09:IBM3101_1X:01:IBM3101_2X:01:TV1920:05
               &+:TV1950:05:VT52:01:VT100:01:[e user term_type]:XX:
           &+ :[e user term_type]:   ]
       &+ :]
&-
&- Wake up Renex converter.
&-
send \015
&-
&- If the terminal number is XX, display the Renex
&- terminal menu and let the user decide what to do.
&-
&if &[e equal &(renex_type) XX] &then &quit
&else &do
&-
&- User terminal type is known.  Discard the Renex menu
&- and tell the Renex converter what the terminal is.
&-
   ..file_output [pd]>&!
   wait "ENTER TERMINAL NUMBER, 2 DIGITS." -time 20
   ..ro; dl [pd]>&!
   send &(renex_type)
&end
&quit
```

APPENDIX B

MULTICS COMMUNICATION SYSTEM MEMORY CONFIGURATOR

This appendix provides a Multics Communication System memory configurator which can be used to approximate maximum memory utilization on the FNP.

These calculations assume an FNP configured with at least 64K of memory.

DN6670 CONFIGURED WITH AT LEAST 64K OF MEMORY

1. Table 1 lists those modules that are required to be in the core image. Also included is the memory required to support the IOM table and interrupt vectors. The init module is released for buffer space at the end of FNP initialization and is not to be included in the total count. The FNP requires around 30 buffers for DIA queues and other support operations and this is included in the minimum buffer pad entry.

TABLE 1

MODULE	LENGTH	TOTAL
control_tables	1828	1828
hsla_man	4602	4602
dia_man	3776	3776
interpreter	2140	2140
scheduler	1310	1310
utilities	3416	3416
minimum buffer pad	960	960
interrupt vect	512	512
iom tables	32	32
paging windows	512	512
init	2772	____
	SubTotal1	19088

2. Table 2 lists the modules that a site may optionally configure based on the CDT and other site requirements. Refer to Section 6 of this manual for details of when one of these should be included. Sum the TOTAL column and enter result in the box after SubTotal2.

TABLE 2

MODULE	LENGTH	TOTAL
acu_tables	128	
* autobaud_tables	282	
breakpoint_man	224	
bsc_tables	2440	
console_man	480	
gl15_tables	1960	
ibm3270_tables	650	
meters	122	
ic_sampler	2104	
polled_vip_tables	1384	
| hasp_tables	1100	
x25_tables	4098	
trace	254	
vip_tables	532	
	SubTotal2	

3. Table 3 determines the amount of memory in the low-order 32K required to support the various types of channels and devices. This space is made up of TIB table entries and TIB extensions. Echoplexed channels for this discussion are those which use any one of the following modes: echoplex, crecho, lfecho, or tabecho. A blank is provided in the table to enter the number of each type of channel or device. Multiply this number by the number in the third column and place result in fourth column. Sum the entries in the "memory required" column and enter result in the box after SubTotal3. Additional space is not required for metering in an extended memory configuration.

The following abbreviations are used below:

```
TIB = terminal information block
SFCM = software communications region
HSLA = high speed line adapter
```

TABLE 3

type of channel	number channels	X	memory/ channel	=	memory required
tty			2		
gl15 protocol			18		
polled_vip protocol			26		
ibm2780 protocol			34		
ibm3780 protocol			34		
HASP protocol			34		
X.25 protocol			36		
HSLA			97		
			SubTotal3		

Calculate the sum of SubTotal1, SubTotal2, and SubTotal3. If it is more than 32768, the FNP is over-configured.

4. Table 4 determines how much space is required for I/O buffers. This space is allocated in extended memory. Echoplexed channels for this discussion are those which use any one of the following modes: echoplex, crecho, lfecho, or tabecho. A blank is provided in the table to enter the number of each type of channel or device. Multiply this number by the number in the third column and place result in fourth column. Sum the entries in the "memory required" column and enter result in the box after SubTotal4.

TABLE 4

type of channel	number channels	X	memory/ channel	=	memory required
non_echoplexed tty			64		
echoplexed tty			96		
gl15 protocol			438		
polled_vip protocol			See Note 1		
ibm2780 protocol			224		
ibm3780 protocol			256		
HASP protocol			See Note 2		
X.25 protocol			See Note 3		
			SubTotal4		

NOTE 1: The size in words of one VIP7760 I/O buffer is controlled by the max_message_len specification in the TTF entry for the multiplexer. The following accounts for three buffers: one output and two input.

```
(max_message_len/2 + 2)*3
```

Enter the above result into the table.

NOTE 2: The size in words of one HASP I/O buffer is expressed by

```
2 + block_size/2
```

where there are two words of buffer overhead and block_size is specified by the TTF entry of the multiplexer. Enter the above result into the table.

NOTE 3: The space required is a function of frame_size and the behavior of the protocol. The maximum space required is:

```
(frame_size/16)*(K+10)
```

but the average is less than (frame_size/16)*K

5. The TIB and SFCM for each channel are allocated in a single 256-word page. Meters for the channel, if metering is enabled, are allocated in the same page, as are the permanent input buffers for asynchronous channels. The remainder of the page, if any, is included in the buffer pool. However, for safety and simplicity, the entire page should be counted against the channel. The space taken up for TIBs and SFCMs is therefore calculated as follows:

```
number of channels x 256 = SubTotal5
```

	256	

6. The maximum size of the trace buffer is determined as follows: calculate the sum of SubTotal4 and SubTotal5 and add 32768. Subtract the result from the total amount of memory configured. This result (or something less) can be used after the size key-word for the trace module in the bindfile for the FNP core image.

APPENDIX C

SPACE REQUIREMENTS IN TTY_BUF

Two terms are defined to help make clear the data that will be presented later in this section: multiplexer channel—a channel that represents a concentrator controlling one or more channels; subchannel—a channel that represents the user's terminal or end device. All channels that are not multiplexer channels are considered to be subchannels.

An FNP is a multiplexer to which, for example, a VIP7801 can be attached. The full name of the channel would be something like "b.h024". The parent multiplexer name is "b" (the FNP), and the subchannel name is "h024".

DATA BASES IN tty_buf

The space used by tty_buf is occupied by various data bases that are used to maintain proper control of the Multics Communication Management.

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 protocol makes different demands on tty_buf space. (See "Dynamic Storage in tty_buf" below.)

Subchannel and Multiplexer Channel Static Data Requirements

FNP subchannels--For those subchannels of the FNP matching the star name *.*, the following is required:

```
PCB       8
LCTE     32
WTCB     20
Total    60 words
```

VIP7760 multiplexer--This multiplexer type requires a data base in tty_buf defined by the "pvmd" structure in the polled_vip_mpx_data.incl.pl1 include file.

```
PCB       8
LCTE     32
PVMD     48
Total    88 words
```

VIP7760 subchannel--Each subchannel of a VIP7760 type multiplexer requires a data base in tty_buf defined by the "pvste" structure in the polled_vip_mpx_data.incl.pl1 include file.

```
LCTE     32
PVSTE     4
WTCB     20
Total    56 words
```

IBM3270 multiplexer--This multiplexer type requires a data base in tty_buf defined by the "md" structure in the ibm3270_mpx_data.incl.pl1 include file.

```
PCB       8
LCTE     32
MD       64
Total   104 words
```

IBM3270 subchannel--Each subchannel of an IBM3270 multiplexer requires a data base in tty_buf defined by the "mde" structure in the ibm3270_mpx_data.incl.pl1 include file.

```
LCTE      32
MDE       12
WTCB      20
Total     64 words
```

X.25 multiplexer—This multiplexer type requires a data base defined by the "x25d" structure in the x25_data.incl.pl1 include file.

```
LCTE     32
X25D     32
Total    64 words
```

X.25 subchannel—Each X.25 subchannel of an X.25 multiplexer requires a data base defined by the "xsce" structure in the x25_data.incl.pl1 include file.

```
LCTE     32
XSCE     14
WTCB     20
Total    66 words
```

X.25 logical channel—Each logical channel of an X.25 multiplexer (defined by the n_lc parameter) requeues a data base defined by the "xlce" structure in x25_data.incl.pl1 include file.

```
XLCE      28
Total     28
```

HASP multiplexer—This 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
```

HASP subchannel—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		
IBM3270 multiplexers			104		
IBM3270 subchannels			64		
VIP7760 multiplexers			88		
VIP7760 subchannels			56		
X.25 multiplexers			64		
X.25 subchannels			66		
X.25 logical channels			28		
			Total		

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.

Some comments can be made about each 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.

Buffer Size When Controlled by Baud Rate

The output buffer size is sometimes controlled by the baud rate of the channel. This occurs as noted in the following sections.

Whether a given block of data is delivered all at once depends on the available space in ring 0 and in the FNP.

At speeds of 4800 baud and below, no more than 960 characters are sent to the FNP at a time. However, you can adjust the baud rate FNP entry in the CMF (see "FNP Entries in the CMF"). When required, use the following equations to determine the size of the buffer.

If the baud rate of the channel is below 1200 baud, assume it is 1200 baud for the following calculations. Each buffer has a one-word header containing the address of the next buffer and the number of characters in the buffer.

```
buffer_char_size = min((64 * baud/1200)-4, 508)
```

The following equation expresses the size of the buffer in words:

```
buffer_word_size = buffer_char_size/4
```

ASYNCHRONOUS I/O BUFFER SPACE

Input buffer size is 16 words when channel is not operating in block transfer mode. If the channel is operating in block transfer mode, the buffers will be based on the baud rate of the channel. Output buffer size is based on the baud rate of the channel (see discussion above). The following formula accounts for one input buffer and one output buffer:

```
if not in block transfer mode;
     ASYNC-IO = buffer_word_size + 16

if in block transfer mode;
     ASYNC-IO = buffer_word_size * 2
```

G115 I/O BUFFER SPACE

Entry in table accounts for two buffers.

HASP I/O BUFFER SPACE

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-IO = 3 * ttf_block_size
```

IBM2780 AND IBM3780 I/O BUFFER SPACE

The size of a buffer is controlled by the value following the -size control argument in the attach description to bisync_. The following formula accounts for two buffers:

```
IBM2780/3780-IO = ((block_size/4) + 1)*2
```

IBM3270 I/O BUFFER SPACE

Entry in table accounts for two buffers.

VIP7760 I/O BUFFER SPACE

The buffer size is controlled by the max_message_len specification in the TTF entry for the multiplexer. The following accounts for two buffers:

```
VIP7760-IO = (max_messsage_len/4 + 1)*2
```

X.25 I/O BUFFER SPACE

Data is transmitted and received in frames. Each frame occupies as many buffers as necessary to hold the frame. The size of the frame is controlled by the frame_size parameter specified in the TTF entry of the multiplexer.

Buffer size for X.25 multiplexers is based on the baud rate of the channel (see above). The frame_size parameter in the TTF entry is used to determine the number of buffers that will hold the frame.

```
buffers_per_frame = (frame_size/8)/buffer_char_size
(to the next whole number)
```

Now the number of words used by each frame can be calculated.

```
frame_words = ((buffer_char_size/4)+1)*buffers_per_frame
```

The X.25 protocol allows up to 7 frames to be transmitted before an acknowledgment needs to be received to send more frames. This is controlled by the window_size parameter in the TTF entry of the multiplexer. It is possible that there would be one output window waiting to be acknowledged plus two output frames being filled in by the multiplexer. Two input frames should also be allowed for. The total number of words for each X.25 multiplexer will be:

```
X25-IO = (window_size + 4)*frame_words
```

Calculation of Dynamic Storage in tty_buf

type of channel	number X	memory/ channel	= memory required
Asynchronous I/O buffer space		ASYNC-IO	
G115 I/O buffer space		170	
HASP I/O buffer space		HASP-IO	
IBM2780/3780 I/O buffer space		IBM2780/3780-IO	
IBM3270 I/O buffer space		130	
VIP7760 I/O buffer space		VIP7760-IO	
X.25 I/O buffer space		X.25-IO	
		Total	

INDEX

V

W

X

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: MULTICS
COMMUNICATIONS ADMINISTRATION

ORDER NO. CC75-02

DATED FEBRUARY 1985

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

FOLD ALONG LINE

FOLD ALONG LINE

CUT ALONG LINE

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy**: 32 Via Pirelli, 20124 Milano
Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F. **Japan**: 2-2 Kanda Jimbo-cho, Chiyoda-ku, Tokyo
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia**: Mandarin Plaza, Tsimshatsui East, H.K.

42370, 7.5C385, Printed in U.S.A.

CC75-02