HONEYWELL

LEVEL 68 MULTICS PROGRAMMERS' MANUAL — COMMUNICATIONS INPUT/OUTPUT

SOFTWARE

SERIES 60 (LEVEL 68)

MULTICS PROGRAMMERS' MANUAL — COMMUNICATIONS INPUT/OUTPUT

SUBJECT

Communications Input/Output Reference Material, Including Command, I/O Module, and Subroutine Descriptions

SPECIAL INSTRUCTIONS

This manual is one of six manuals that constitute the *Multics Programmers' Manual* (MPM).

Reference Guide	Order No. AG91
Commands and Active Functions	Order No. AG92
Subroutines	Order No. AG93
Subsystem Writer's Guide	Order No. AK92
Peripheral Input/Output	Order No. AX49
Communications Input/Output	Order No. CC92

This revision supersedes Revision 0 of the manual dated October 1978 and Addendum A dated November 1979. Change bars indicate new and changed information.

SOFTWARE SUPPORTED

Multics Software Release 9.0

ORDER NUMBER

CC92-01

August 1981

Honeywell

PREFACE

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

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

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

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

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

File No.: 1L13,1L63

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

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

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

Examples of specialized subsystems for which construction would require reference to the MPM Subsystem Writers' Guide are:

- A subsystem that precisely imitates the command environment of some system other than Multics.
- A subsystem intended to enforce restrictions on the services available to a set of users (e.g., an APL-only subsystem for use in an academic class).
- A subsystem that protects some kind of information in a way not easily expressible with ordinary access control lists (e.g., a proprietary linear programming system, or an administrative data base system that permits access only to program-defined, aggregated information such as averages and correlations).

Several cross-reference facilities help locate information:

- Each manual has a table of contents that identifies the material (either the name of the section and subsection or an alphabetically ordered list of command and subroutine names) by page number.
- Each manual contains an index that lists items by name and page number.

Portions of this manual give information most useful for special applications of the Multics Communication System. These sections are of limited interest to general users, and include: "Syntax of the TTF" in Section 3, the ttt_info_ subroutine described in Section 5, and the I/O modules, except tty_, described in Section 6.

One additional manual referenced is the Multics Administrators' Manual--Communications, Order No. CC75. It is referred to in the text as MAM Communications.

Changes to MPM Communications I/O contained in Addendum A include: new baud rate information in Section 3; a new command, the dial_out command, in Section 4; a number of changes and clarifications to the tty_ I/O module description in Section 6, including the new control arguments -dial_id and -resource; and changes to the printer modes described in Appendix B.

CONTENTS

		Page
Section 1	Overview of Multics Communication System . . .	1-1
	Terminals and Channels	1-1
	Attachments	1-1
	Data Transformation	1-2
Section 2	Use of Terminals on Multics	2-1
	ASCII Character Set	2-1
	Printing Graphic Characters	2-1
	Control Characters	2-3
	Nonstandard Control Character	2-4
	Unused Characters	2-4
	Typing Conventions	2-4
	Canonical Form	2-4
	Canonicalization	2-6
	Column Assignment	2-6
	Overstrike Canonicalization	2-7
	Overstrike Canonicalization Examples . .	2-7
	Replacement Canonicalization	2-8
	Replacement Canonicalization Examples . .	2-9
	Erase and Kill Characters	2-9
	Examples of Erase and Kill Processing . .	2-11
	Escape Sequences	2-11
	Typing Convention Examples	2-12
	Column Canonicalization Examples	2-12
	Erase, Kill, and Escape Examples	2-13
	Terminal Output	2-15
	Carriage Motion	2-16
	Delays	2-17
	Output Escape Sequences	2-17
	Continuation Lines	2-17
	End-of-page Processing	2-18
	Escape Conventions on Various Terminals . .	2-18
	Selectric Devices	2-18
	Upper Case Only Devices	2-19
	Execuport 300	2-19
	CDI Model 1030	2-19
	Flow Control	2-20
	Input Flow Control	2-20
	Output Flow Control	2-20
	Block Transfer	2-20
Section 3	Terminal Types	3-1
	Terminal Type Concept	3-1
	Terminal Type and Line Type	3-1
	Terminal Type Table and Terminal Type File .	3-1
	Setting Terminal Types	3-2
	Changing Terminal Type Definitions . . .	3-2
	Terminal Type Table	3-3
	Syntax of the TTF	3-4
	Generalized Character Specifications . .	3-4
	Terminal Type Entry	3-5
	Video Table Definition	3-10
	Global Statements	3-14
	Conversion Table Entry	3-15
	Translation Table Entry	3-15
	Function Key Table Entry	3-15
	Example	3-16
	Special Characters Table Entry	3-16

CONTENTS (cont)

		Page
	Default Types	3-18
	Answerback Table	3-18
	Preaccess Commands	3-19
	Examples	3-20
Section 4	Commands	4-1
	cv_ttf	4-2
	dial_out	4-2.1
	display_ttt	4-3
	l6_ftf	4-4
	print_terminal_types, ptt	4-6
	print_ttt_path	4-7
	set_ttt_path	4-8
	set_tty, stty	4-9
Section 5	Subroutines	5-1
	ttt_info_	5-2
	ttt_info_$terminal_data	5-2
	ttt_info_$modes	5-5
	ttt_info_$preaccess_type	5-5
	ttt_info_$additional_info	5-6
	ttt_info_$initial_string	5-6
	ttt_info_$dialup_flags	5-7
	ttt_info_$decode_answerback	5-8
	ttt_info_$encode_type	5-8
	ttt_info_$decode_type	5-9
	ttt_info_$video_info	5-10
	ttt_info_$function_key_data	5-14
Section 6	Input/Output Modules	6-1
	bisync_	6-2
	g115_	6-7
	hasp_host_	6-10
	SIGNON Processing	6-15
	hasp_workstation_	6-18
Index		i-1

TABLES

Table 2-1. ASCII Character Set on Multics 2-2

SECTION 1

OVERVIEW OF MULTICS COMMUNICATION SYSTEM

The Multics Communication System (MCS) effects the transfer of data between the Multics virtual memory and various remote devices (primarily terminals) over communications channels. This manual is concerned with MCS as it appears to the user of a terminal. For a description of the internal workings of MCS, see the appropriate program logic manual.

The bulk of MCS resides in the Multics supervisor and in a separate machine, the Front-End Network Processor (FNP). The user-ring and supervisor portions of MCS are principally concerned with terminal management, while the FNP's primary responsibility is channel management. In general, the user need not be concerned with channel management. Most user and system programs interface to MCS through the input/output system by means of the iox_ subroutine, described in the MPM Subroutines. For general information on the use of the I/O system, see "Input and Output Facilities" in the MPM Reference Guide.

TERMINALS AND CHANNELS

The term "channel" (or "communications channel"), as used in this manual, refers to a physical connection between an FNP and a remote input/output device. Such a connection may go through a telephone system or a private communications network, or it may consist of one or more hardwired cables. For information on the specification and management of all communications channels known to the system, see the MAM Communications.

The word "terminal" is used to refer to the device itself; it may be an ordinary interactive terminal on which a user types commands, or it may be a computer controlling one or more peripheral devices.

ATTACHMENTS

An interactive terminal is normally connected to the system (attached) through the tty_ I/O module described in Section 6. For the user's login terminal, this attachment is performed automatically in the course of process creation. Additional terminals connected to the user's process using the dial facility must be attached explicitly. For more information on the dial facility, see the descriptions of the dial command in MPM Commands and the dial_manager_ subroutine in the MPM Subsystem Writer's Guide.

Other types of devices that use special communications protocols may have to be attached through special-purpose I/O modules. Several such modules are supplied with the system; they are described in Section 6. Users and sites may also supply their own I/O modules that interface to one of the existing modules. For information, see "Implementation of Input/Output Modules" in the MPM Subsystem Writer's Guide.

DATA TRANSFORMATION

One of the most visible functions of MCS is the transformation of data read from or written to the terminal. This may include rearrangement of white space, replacement of one character by a sequence of characters, and, in some cases, wholesale translation from one character code to another. The types of conversion for input and output are described in Section 2. The specific details of any particular conversion are determined by terminal type and, to a lesser extent, by the modes associated with the attachment. Terminal types are explained in Section 3, and the effects of the various modes are given in the description of the tty_ I/O module. The set_tty command, described in Section 4, can be used to change the terminal type or to modify many of the parameters used in converting input or output.

The special-purpose I/O modules (those other than tty_) usually perform their own data conversions independent of terminal type. They generally put the terminal in rawi and rawo modes (i.e., "raw" input and output) to prevent the rest of MCS from performing any transformations on data to or from the terminal.

SECTION 2

USE OF TERMINALS ON MULTICS

ASCII CHARACTER SET

The Multics standard character set is the revised U.S. ASCII Standard (refer to USA Standards Institute, "USA Standard X3.4-1968"). The ASCII set consists of 128 7-bit characters. These are stored internally, right-justified, in four 9-bit fields per word. The two high-order bits in each field are expressly reserved for expansion of the character set; no system program may use them. Any hardware device that is unable to accept or create the full character set should use established escape conventions for representing the set (see "Escape Characters" below). There are no meaningful subsets of the revised ASCII character set.

The ASCII character set includes 94 printing graphics, 33 control characters, and the space. Multics conventions assign precise interpretations to all the graphics, the space, and 10 of the control characters. The remaining 23 control characters are presently reserved.

Printing Graphic Characters

The printing graphic characters are the uppercase alphabet, the lowercase alphabet, digits, and a set of special characters. The special characters are listed below.

!	exclamation point	;	semicolon
"	double quote	<	less than
#	number sign	=	equals
$	dollar sign	>	greater than
%	percent	?	question mark
&	ampersand	@	commercial at
'	acute accent	[	left bracket
(	left parenthesis	\	left slant
)	right parenthesis	]	right bracket
*	asterisk	^	circumflex
+	plus	_	underline
,	comma	`	grave accent
-	minus	{	left brace
.	period	¦	vertical bar
/	right slant	}	right brace
:	colon	~	tilde

Note: The solid vertical bar (\|) and the broken vertical bar (¦) are equivalent representations of the graphic corresponding to ASCII code 174.

Table 2-1. ASCII Character Set on Multics

	0	1	2	3	4	5	6	7
000	(NUL)							BEL
010	BS	HT	NL	VT	NP	CR	RRS	BRS
020								
030								
040	Space	!	"	#	$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	^	_
140	`	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	¦	}	~	PAD

Control Characters

The following conventions define the standard meanings of the ASCII control characters that are given precise interpretations in Multics. These conventions are followed by all standard I/O modules and by all system software inside the I/O system interface. Since some devices have different interpretations for some characters, it is the responsibility of the appropriate I/O module to perform the necessary translations.

The characters designated as unused are specifically reserved and can be assigned definitions at any time. Until defined, unused control characters are output using the octal escape convention in normal output and are not printed in edited mode. Users wishing to assign interpretations for an unused character must use a nonstandard I/O module.

If a device does not perform a function implied by a control character, its standard I/O module provides a reasonable interpretation for the character on output. This might be substituting one or more characters for the character in question, printing an octal escape, or ignoring it.

The Multics standard control characters are:

BEL	Sound an audible alarm.
BS	Backspace. Move the carriage back one space. The backspace character implies overstrike rather than erase.
HT	Horizontal tab. Move the carriage to the next horizontal tab stop. Multics standard tab stops are at 11, 21, 31... when the first column is numbered 1.
NL	Newline. Move the carriage to the left end of the next line. This implies a carriage return plus a line feed. ASCII LF (octal 012) is used for this character.
VT	Vertical tab. Move the carriage to the next vertical tab stop and to the left of the page. Standard tab stops are at lines 11, 21, 31... when the first line is numbered 1. This character, by definition, does not appear in a canonical string.
NP	New page. Move the carriage to the top of the next page and to the left of the line. ASCII FF (octal 014) is used for this character.
CR	Carriage return. Move the carriage to the left of the current line. This character, by definition, does not appear in a canonical string.
RRS	Red ribbon shift. ASCII SO (octal 016) is used for this character.
BRS	Black ribbon shift. ASCII SI (octal 017) is used for this character.
PAD	Padding character. This is used to fill out words that contain fewer than four characters and that are not accompanied by character counts. This character cannot appear in a canonical character string. ASCII DEL (octal 177) is used for this character.

Nonstandard Control Character

One control character, NUL, is recognized under certain conditions by all device interface modules because of its wide use outside Multics. This character is handled specially only when the I/O module is printing in edited mode, and is, therefore, ignoring unavailable control functions. The null character is ASCII character NUL (octal 000). In normal mode, this character is printed with an octal escape sequence; in edited mode, it is treated exactly as PAD. This character cannot appear in a canonical character string. Programmers are warned against using NUL as a routine padding character and using edited mode on output because all strings of zeros, including mistakenly uninitialized strings, are discarded.

Unused Characters

These characters are reserved for future use:

SOH	001	ACK	006	DC4	024	SUB	032
STX	002	DLE	020	NAK	025	ESC	033
ETX	003	DC1	021	SYN	026	FS	034
EOT	004	DC2	022	ETB	027	GS	035
ENQ	005	DC3	023	CAN	030	RS	036
				EM	031	US	037

TYPING CONVENTIONS

Three categories of typing conventions are dealt with in this discussion: canonical form, erase and kill characters, and escape characters.

Canonical Form

A character stream is a representation of one or more printed lines. Since the same printed line can be produced using different sets of keystrokes, there are several possible character streams that represent the same line. For example, the line:

```
     start    lda  alpha,4      get first result.
```

could have been typed with either spaces or horizontal tabs separating the fields; one cannot tell by looking at the printed image.

A program should be able to compare two character streams easily to see if they produce the same printed image. It follows that all character input to Multics must be converted into a standard (canonical) form. Similarly, all programs producing character output, including editors, must produce canonical form output streams.

Of all possible ASCII character strings, only certain strings are ever found within Multics. All strings that produce the equivalent printed effect on a terminal are represented within Multics as one string, the canonical form for the printed image. The user, however, is free to type a noncanonical character stream. This stream is automatically converted to the canonical form before it reaches his program. An exception to this automatic conversion is that tab characters are preserved; a detailed description of the conversion process is found later in this section. If the user wants his program to receive raw or partially processed input from his terminal, an escape mechanism is provided by the modes operation of the tty_ I/O module. The I/O module is accessed via a call to the iox_ subroutine (see the description of the iox_ subroutine in the MPM Subroutines). The modes available that apply to canonicalization are:

^can	no canonicalization of overstrikes.
^esc	no canonicalization of escape characters.
^erkl	no erase and kill processing.
rawi	read the specified data from the terminal without any conversion or processing. This includes shift characters and undifferentiated uppercase and lowercase characters.

Similarly, an I/O module is free to rework a canonical stream on output into a different form if, for example, the different form happens to print more rapidly or reliably on the device.

The current Multics canonical form is designed for the convenient typing of aligned tabular information, which requires an ambiguous interpretation of the tab character. The following three statements describe the current Multics canonical form.

1. A text line is a sequence of character positions separated by horizontal carriage motion and ending in a newline character.

2. Carriage motion consists of newline, tab, and space characters.

3. A character position consists of a single graphic or several overstruck graphics. A graphic is a printable character. An overstruck character position consists of two or more graphics separated by backspaces. Regardless of the order in which the graphics are typed, they are always stored in ascending ASCII order. Therefore, the symbol "$\underline{x}$", whether typed as:

```
          >B<B_
     or
          <B>B_
     or
          _B<B>
```

   is always stored internally as:

```
          <B>B_
```

   where B is a backspace.

There are any number of ways to type two or more consecutive overstruck character positions. The graphics in each position are grouped together, so that "$\underline{xx}$" is always stored as:

```
<B>B_<B>B_
```

The following paragraphs give a complete set of rules for transforming a typed line into the form in which it is stored, followed by further examples illustrating the rules. The transformation process is carried out in three steps: canonicalization, erase/kill processing, and escape processing. If two or more of the rules listed below are applicable to a given input string, they are applied in the order in which they are presented here.

Canonicalization

Canonicalization is the process of converting an input string into canonical form. Two methods of canonicalization are defined on Multics: a method for printing terminals and a method for video (CRT) terminals. Both methods of canonicalization attempt to ensure that what is visible on the terminal is the canonical form of the input string. The method used is determined by the setting of the "can_type" mode, as explained in the description of the tty_ I/O module elsewhere in this manual.

Canonicalization for printing terminals (overstrike canonicalization) is designed for terminals which are capable of overstriking multiple characters in a single column. Any group of overstruck characters is converted to a single representation regardless of the order in which the characters were entered into the column.

Canonicalization for video terminals (replacement canonicalization) is designed for terminals which are not capable of overstriking. When a character is entered into a column, any characters previously present in that column are no longer visible. Replacement canonicalization mimics this behavior of the terminal by only placing the last character typed into any column into the canonical representation of the string.

The canonicalization process consists of two distinct steps: column assignment, which is identical for both methods of canonicalization, and the actual canonicalization process.

Column Assignment

The following rules are used to determine which printing graphics, if any, appear in each physical column position.

1. The leftmost position of the carriage is considered to be column 1.
2. Each printing graphic or space typed increases the column position by 1.
3. Each backspace typed decreases the column position by 1 unless the column position is 1.
4. A carriage return sets the column position to 1.
5. A horizontal tab increases the column position to the next tab stop; tab stops are defined to be at columns 11, 21, 31, etc.
6. A newline, formfeed, or vertical tab sets the column position to 1 and advances the carriage vertically; thus no character typed after such a character can share a column position with a character typed before it.

7. If the terminal is not in ctl_char mode, any ASCII control character other than backspace, horizontal tab, newline, vertical tab, formfeed, and carriage return is discarded. If the terminal is in ctl_char mode, such characters are treated as if they were printing graphics (with the exception of the NUL character, which is always discarded). The default is that ctl_char mode is off.

Overstrike Canonicalization

The following rules determine the formation of the canonical string.

1. Characters on each line are sorted so that their associated column positions are monotonically increasing.
2. No carriage return characters may appear in the canonical string.
3. A horizontal tab is preserved as typed unless a printing graphic appears in one of the columns skipped by the tab, in which case the tab is replaced by an appropriate number of spaces.
4. Backspaces appear in the canonical string only when two or more printing graphics share a column position.
5. When two or more different printing graphics share a column position, the characters are sorted as follows: graphic with lowest numeric ASCII code, backspace, graphic with next lowest numeric ASCII code, etc.
6. If the contents of a column position consist of two or more instances of the same printing graphic, that column is reduced to a single instance of the graphic.
7. A line-ending character (newline, formfeed, or vertical tab) immediately follows the last printing graphic in the rightmost column position on the line.

Overstrike Canonicalization Examples

Several illustrations of canonical form are shown below. Assume that the typist's terminal has horizontal tab stops set at 11, 21, 31, etc.

```
Typist:         this is ordinary text.N
Typed line:     this is ordinary text.
Canonical form: this is ordinary text.N
```

where N is the newline character. In most cases, the canonical form is the same as the original key strokes of the typist, as above.

```
Typist:         here fullBBBB____ means thatN
Typed line:     here full means that
Canonical form: here _Bf_Bu_Bl_Bl means thatN
```

where B is a backspace and N is a newline character. This is the most common type of canonical conversion, done to ensure that overstruck graphics are stored in a standard pattern.

```
Typist:         We see no probSBlemC__N
Typed line:     We see no problem
Canonical form: WB_Be see no problemN
```

where B is a backspace, N is a newline character, S is a space, and C is a carriage return. The space between "prob" and "lem" was not overstruck; it and the following backspace were simply removed. Note the difference in the storage of the characters that were overstruck in this and the preceding example; the ASCII code value of the underscore is between the values for uppercase and lowercase letters.

Replacement Canonicalization

Replacement canonicalization is designed for use on a terminal with the following characteristics:

- Overstriking a character with any other printing character or a space causes the first character to be erased.
- Entering a tab character simply moves the cursor position to the next tab stop (column 11, 21, etc.) without erasing any intervening characters.

The following rules determine the formation of the canonical string:

1. Characters on each line are sorted so that their associated column positions are monotonically increasing.
2. No carriage return characters may appear in the canonical string.
3. A horizontal tab is preserved as typed unless a printing graphic appears in one of the columns skipped by the tab, in which case the tab is replaced by an appropriate number of spaces.
4. When two or more characters (including space and identical printing graphics) share a column position, the last character entered by the user in that column is kept and all other characters in that column discarded.
5. A line-ending character (newline, formfeed, or vertical tab) immediately follows the last printing graphic in the rightmost column position on the line.

With replacement canonicalization, as seen above, it is not possible to overstrike two characters, as the last one typed is always the only character in that column. Thus it is not possible to use the feature of overstriking a character with the erase character, as described in the "Erase and Kill Characters" section following, to delete a character typed in the middle of a line. Instead, to delete such a character, you must reposition to the character in question and retype the remainder of the line being input.

Therefore, you may want to disable the erase character when using replacement canonicalization. This may be accomplished by the command line:

```
set_tty -edit \400
```

where \400 is a character which cannot normally be entered on the terminal.

Replacement Canonicalization Examples

Several illustrations of canonical form are shown below. Assume that the typist's terminal has horizontal tab stops set at 11, 21, 31, etc.

```
Typist:          this is ordinary text.N
Screen contents: this is ordinary text.
Canonical form:  this is ordinary text.N
```

where N is the newline character. In most cases, the canonical form is the same as the original key strokes of the typist, as above.

```
Typist:          this is a msitake.BBBBBBBisN
Screen contents: this is a mistake.
Canonical form:  this is a mistake.N
```

where B is a backspace and N is a newline character. This example illustrates the correction of errors in the middle of a typed line. It is the most common use of replacement canonicalization.

```
Typist:          this si a strange BBBBBBBBBBBBBisHHBBexample.N
Screen contents: this is a strange example.
Canonical form:  this is a strange example.N
```

where B is a backspace, H is a horizontal tab, and N is a newline character. This example illustrates that the horizontal tab character does not erase intervening characters (" a strange" in this example).

```
Typist:          This is   some text.BBBBBBBBBBBsome text.    N
Screen contents: This is some text.
Canonical form:  This is some text.N
```

where B is a backspace and N is a newline character. This example illustrates that in order to erase extra whitespace in a line, the typist must position to the first extraneous character, retype the remainder of the line, and type sufficient spaces at the end of the line to overstrike any extra undesired characters.

If, in the above example, the final spaces are not typed, the following occurs:

```
Typist:          This is   some text.BBBBBBBBBBBsome text.N
Screen contents: This is some text.t.
Canonical form:  This is some text.t.N
```

Erase and Kill Characters

Two capabilities for minimally editing the line being typed are available. They are:

- The ability to delete the latest character or characters (erase)
- The ability to delete all of the current line (kill)

By applying canonical form to these two editing functions, it is possible to interpret unambiguously a typed line in which editing was required.

The first editing convention reserves one graphic as the erase character. On Multics, the default erase character is the number sign (#). The user can designate a different character by invoking the set_tty command with the -edit control argument. Although the erase character is a printed graphic, it does not become part of the line. When it is the only graphic in a print position, it erases itself and the contents of the previous print position. Several successive erase characters erase an equal number of print positions. One erase character typed immediately after "white space" causes the entire white space to be erased (any combination of tabs and spaces is called white space). The number sign can be struck over another graphic. In this case it erases the print position on which it appears. For example, typing:

```
     theSSne###next
  or
     theST#next
  or
     the#next
```

where S is a space and T is a horizontal tab, produces:

```
     thenext
```

Since processing of erase characters takes place after the transformation to canonical form, there is no ambiguity as to which graphic character has been erased. The printed image is always the guide.

The second editing convention reserves another graphic as the kill character. On Multics, the default kill character is the commercial at sign (@). Again, the user can redesignate this. When this character is the only graphic in a print position, the contents of that line up to and including the kill character are discarded. Again, since kill processing occurs after the conversion to canonical form, there is no ambiguity about which characters have been discarded.

By convention, an overstruck erase character is processed before a kill character, and a kill character is processed before a nonoverstruck erase character. Therefore, the only way to erase a kill character is to overstrike it with an erase character.

Because of their special meanings to Multics, these two graphics should be avoided in software.

The following rules apply to erase and kill characters.

1. If the terminal is in esc mode, an erase or kill character alone in a column immediately preceded by an escape character alone in a column is not processed as an erase or kill character.

2. An erase character alone in a column position and preceded by more than one blank column results in the deletion of <u>all</u> immediately preceding blank columns, as well as of the erase character.

3. An erase character alone in a column position results in the deletion of itself and of the contents of the preceding column position.

4. An erase character sharing a column position with one or more printing graphics results in the deletion of the contents of that column position.

5. A kill character results in the deletion of its own column position and all column positions to its left, unless it shares a column position with an erase character, in which case rule 4 applies (the kill character is erased).

Notice that for rule number 1 to apply, the erase or kill character must actually have been typed in the column immediately following the escape character. The reason for this is that it facilitates the erasing of escape sequences, e.g., \001####.

Examples of Erase and Kill Processing

```
Typist:          abcx#deSBfzz##gN
Typed line:      abcx#defzz##g
Canonical form:  abcx#defzz##gN
Final input:     abcdefgN


Typist:          this@In the offBBB___##nB_ stateN
Typed line:      this@In the off##n state
Canonical form:  In the _Bo_Bn stateN
Final input:     In the on state
```

Escape Sequences

Some terminals cannot print all 128 ASCII characters. To maintain generality and flexibility, standard software escape conventions are used for all terminals. Each class of terminal has a particular character assigned to be the software escape sequence character in the terminal type file. When this character occurs in an input (or output) string to (or from) a terminal, the next character (or characters) are interpreted according to the conventions described below. The escape sequence character should not be confused with the ASCII ESC, which is octal 033.

The standard escape sequence character in Multics is the left slant (\); like the erase and kill characters, it should be avoided in Multics software. The universal escape conventions are:

1. The string \d1d2d3 represents the octal code d1 d2 d3 where d*i* is a digit from zero to seven. Any arbitrary character can be represented this way. The string \d2d3 is equivalent to \d1d2d3 if d1 is zero. The string \d3 is equivalent to \d1d2d3 if d1 and d2 are zero.

2. Local (i.e., concealed) use of the newline character that does not go into the computer-stored string on input, and is not in the computer-stored string on output, is effected by typing \<newline character>.

3. The characters \# place an erase character into the input string.

4. The characters \@ place a kill character into the input string.

5. The characters \\ place a left slant character into the input string.

The escape conventions described in items 1 through 5 above apply only if none of the characters involved are overstruck.

The following rules apply to escape sequences.

1. An escape sequence consists of an escape sequence character alone in its column position followed by one or more printing graphics each of which is alone in its column position. An escape sequence is replaced by a single character in the canonical string.

2. An escape sequence consisting of two successive escape sequence characters is replaced by an escape sequence character.

3. An escape sequence consisting of an escape sequence character followed by an erase or kill character is replaced by an erase or kill character.

4. An escape sequence consisting of an escape sequence character followed by one, two, or three octal digits is replaced by the character whose ASCII value is represented by the sequence of octal digits.

5. An escape sequence character followed by a newline character results in the deletion of both characters from the canonical string.

6. Other escape sequences may be defined on a per-terminal-type basis, where such a sequence consists of an escape sequence character and one character following.

7. If the character following an escape sequence character does not result in an escape sequence as defined by the six rules above, the escape sequence character and following characters are stored as they appear on the line.

TYPING CONVENTION EXAMPLES

In the examples below, the following conventions are used:

N	represents a newline
C	represents a carriage return, assuming that the mode lfecho is not set
B	represents a backspace
T	represents a horizontal tab
S	represents a space
{nnn}	represents a character whose ASCII value is nnn (octal)
\	is the escape sequence character
#	is the erase character
@	is the kill character

The examples in the first group illustrate how various typed sequences are canonicalized in terms of column position; these are followed by examples of erase, kill, and escape canonicalization. In the second group, lines are shown as they appear physically, with no consideration given to the precise sequence of keystrokes that might have produced them.

Column Canonicalization Examples

```
Typed:      nothing special about this line.N

Appearance: nothing special about this line.

Result:     nothing special about this line.N
```

```
Typed:      extraneous white sSBpace is ignored.CSN

Appearance: extraneous white space is ignored.

Result:     extraneous white space is ignored.N


Typed:      Here are two ways (2B_) to overstrike.C____N

Appearance: Here are two ways (2) to overstrike.

Result:     HB__Be_Br_Be are two ways (2B_) to overstrike.N


Typed:      tab + backspace isTBreduced to spaces.N

Appearance: tab + backspace is    reduced to spaces.

Result:     tab + backspace isSSSSreduced to spaces.N
```

NOTE: See rule 3 under "Formation of the Canonical String" above.

Erase, Kill, and Escape Examples

The first few examples illustrate erase and kill processing; the remaining examples illustrate both escape processing and erase and kill processing. These examples assume the terminal is in esc mode (mentioned in rule 1 under "Erase and Kill Characters" and described in the tty_ I/O module) and that overstrike canonicalization is being used.

```
Typed:      abz#cde

Appearance: abz#cde

Result:     abcde


Typed:      abSSS#cde

Appearance: ab   #cde

Result:     abcde


Typed:      not@neverSobB#nSMonday.

Appearance: not@never o#n Monday.

Result:     never on Monday.


Typed:      nox#wBBBBB___S_Sit'sSright.

Appearance: nox#w it's right.

Result:     now it's right.
```

Typed: noxBBB____B#wB_Sit'sSright.

Appearance: nox#w it's right.

Result: noxw it's right.

NOTE: Erase character is overstruck; see rule 4 under "Erase and Kill Characters" above.

Typed: dclSrrsScharS(1)SstaticSinit("\017#6");

Appearance: dcl rrs char (1) static init("\017#6");

Result: dcl rrs char (1) static init("{016}");

Typed: \023B_

Appearance: \023

Result: {002}3

NOTE: Overstruck 3 is not part of escape sequence.

Typed: \B_112

Appearance: \112

Result: \112

NOTE: Overstruck \ is not an escape character.

Typed: a\##b

Appearance: a\##b

Result: a\b

NOTE: According to rule 1 under "Erase and Kill Characters," the first # is not an erase character; according to rule 3 under "Erase and Kill Characters," the second # erases itself and the preceding #.

Typed: a\@#b

Appearance: a\@#b

Result: a\b

NOTE: Same note as in immediately preceding example.

Typed: a\B#@b

Appearance: a#@b

Result: b

NOTE: The \ is erased by the overstruck #.

Typed: a\\#b

Appearance: a\\#b

Result: a\#b

NOTE: According to rule 1 under "Erase and Kill Characters," erase canonicalization does not recognize the #; according to rule 2 under "Escape Sequences," escape canonicalization recognizes \\ and attaches no special meaning to the #.

Typed: a\\##b

Appearance: a\\##b

Result: a\b

NOTE: According to rules 1 and 3 respectively under "Erase and Kill Characters," the first # is not an erase character and the second # erases itself and the preceding #; according to rule 2 under "Escape Sequences," \\ reduces to \.

Typed: a\\###b

Appearance: a\\###b

Result: a\b

NOTE: The first # is not an erase character; the next two are, erasing the second \ and the first #.

Typed: a\\####b

Appearance: a\\####b

Result: ab

NOTE: The first # is not an erase character, and must be erased before the two \ characters. The previous examples illustrate the difficulty of erasing a double \; the clearest method is probably to overstrike (a##b).

Typed: a¢<#b (typed on an IBM Model 2741-like terminal)

Appearance: a¢<#b

Result: a\b

NOTE: Only the < is erased; ¢ is translated to \ (see "Escape Conventions on Various Terminals" below).

TERMINAL OUTPUT

Certain transformations are performed on output destined for a terminal to ensure that it is displayed correctly. These transformations can be broken down into the following categories: carriage motion, delays, escape sequences, continuation lines, and end-of-page processing.

Carriage Motion

Six entries in the terminal's special characters table specify the character sequences to be output when any of the various carriage motion (space, formfeed, vertical tab, horizontal tab, backspace, carriage return, and newline) characters are encountered (for information on this table, see the description of the set_special order to the tty_ I/O module). The most usual case is that the sequence for newline consists of carriage return followed by newline (i.e., linefeed), and each of the other sequences either consists of the source character itself or is null to indicate that the specified function is not available.

In general, carriage motion is reduced to its simplest and most efficient form. Any combination of consecutive carriage motion characters is output as net right or left motion, e.g.:

```
SSBSS
```

is output as:

```
SSS
```

where S is a space and B is a backspace. If a newline immediately follows other carriage motion characters, those carriage motion characters are omitted. In addition, a combination of spaces and horizontal tabs that moves the carriage to or over a tab stop is converted to tabs followed by the minimum possible number of spaces. Tab stops are assumed to be at columns 11, 21, 31, etc. Thus the following sequence (starting at column 1):

```
abcdSSSSSSSSSef
```

is converted to:

```
abcdTSSSef
```

where S is a space and T is a horizontal tab. An exception arises if the terminal is in ^tabs mode or if the special characters table specifies a zero-length sequence for horizontal tabs. In either of these cases, all rightward carriage motion is output as spaces; as many spaces are output as necessary to reach the appropriate column position.

Net left carriage motion is normally output as backspaces unless the final column position is so near the left margin that it is more efficient to output a carriage return followed by spaces. Thus:

```
abcdefgCSSSS___
```

is output as:

```
abcdefgBBB___
```

whereas:

```
abcdefghijkBBBBBBBBB__
```

is output as:

```
abcdefghijkCSS__
```

where C is a carriage return, S is a space, and B is a backspace.

If the terminal lacks the capability to perform a carriage return without a linefeed, the carriage return sequence in the special characters table should be null, in which case net left carriage motion is always output as backspaces. Conversely, if the terminal lacks the backspace function, the backspace sequence should be null, and all net left carriage motion is output as a carriage return followed by spaces. If both sequences are null, net left carriage motion is ignored.

Delays

Printing terminals frequently require more than one character time to move the carriage in any way other than one position to the right. In order to allow the terminal time to reach the column position in which it is next supposed to print, MCS may output one or more ASCII NUL characters following a carriage motion character. NUL characters used in this way are called delays.

The number of delays required in any given situation depends on the terminal mechanism, the distance the carriage has to travel, and the speed at which characters are sent to the terminal (baud rate). The delay table (described under the set_delay order to the tty_ I/O module) contains values, appropriate to the particular terminal and baud rate, that determine the number of delays required for any carriage motion character causing a number of columns to be traversed. The terminal type file (TTF), described in Section 3, contains a specification of delay tables to be used at various speeds for each terminal type. To construct a new terminal type entry, it may be necessary to obtain formulas from the terminal manufacturer from which the necessary delay table values can be derived.

Output Escape Sequences

A character that a particular terminal is incapable of printing may be represented by an escape sequence. The substitution of an escape sequence for a particular character is dictated by that character's entry in the output conversion table (described under the set_output_conversion order to the tty_ I/O module). Two kinds of escape sequences are defined: octal escape sequences, and special escape sequences. An octal escape sequence, as explained earlier, consists of a left slant character followed by three octal digits representing the ASCII value of the character being replaced (e.g.,\012). A special escape sequence is one specified in the special characters table, and consists of zero to three arbitrary characters. Each special escape sequence has two forms, one used in edited mode and one used in ^edited mode. See the descriptions of the set_output_conversion order, the set_special order, and edited mode for the tty_ I/O module for more detailed information.

Continuation Lines

When the length of an output line (i.e., the number of column positions between two newline characters) exceeds the terminal's physical paper or screen width, a newline sequence is inserted and the excess characters appear on the following line, preceded by a continuation sequence consisting of the characters \c . A "line" of arbitrary length can be output using as many continuation lines as necessary. The physical line length of the terminal is made available to the software by means of the line length (ll) mode of the tty_ I/O module.

End-of-page Processing

The page length (pl) mode of the tty_ I/O module may be used to specify the physical length in lines of a page. This feature is primarily of interest to users of video display terminals as a means of preventing output from being scrolled off the screen before it can be read. If page-length checking is enabled, then the last line of a page contains a warning string consisting of the end-of-page sequence specified in the output conversion table (described under the set_output_conversion order to the tty_ I/O module); this sequence is normally the characters "EOP". The output stops when the page is full, and restarts when the user types a newline or formfeed character. If the end-of-page sequence is a null string, output stops at the right margin of the last line of the page, and no warning string is displayed. See the descriptions of pl and scroll modes for further information.

ESCAPE CONVENTIONS ON VARIOUS TERMINALS

The following paragraphs list escape conventions for some of the terminals that can be used to access the Multics system. In general, the conventions described here apply to logging in and out as well as to all other typing. For user convenience, terminals should support the full (128 characters) ASCII character set on input and output. For terminals that do not have a full ASCII character set, escape conventions have been provided. Any of these escape conventions, however, can be respecified by the user.

Selectric Devices

Each typeball used requires a different set of escape conventions.

With the EBCD typeball number 963, the following non-ASCII graphics are considered to be stylized versions of ASCII characters:

¢	(cent sign)	for	\	(left slant, software escape)
'	(apostrophe)	for	´	(acute accent)
¬	(negation)	for	^	(circumflex)

The following escape conventions have been chosen to represent the remainder of the ASCII graphics.

¢'	for	`	(grave accent)
¢<	for	[	(left bracket)
¢>	for	]	(right bracket)
¢(	for	{	(left brace)
¢)	for	}	(right brace)
¢t	for	~	(tilde)

With the correspondence typeball number 029, the following non-ASCII graphics are considered to be stylized versions of ASCII characters.

¢	(cent sign)	for	\	(left slant)
'	(apostrophe)	for	´	(acute accent)
±	(plus-minus)	for	^	(circumflex)

The following escape conventions have been chosen to represent the remainder of the ASCII graphics.

¢(	for	<	(less then)
¢)	for	>	(greater than)
¢l	for	[	(left bracket)
¢r	for	]	(right bracket)
¢:	for	!	(exclamation point)
¢t	for	~	(tilde)
¢'	for	`	(grave accent)
¢/	for	¦	(vertical bar)

NOTE: The left and right braces ({ and }) must be input using octal escapes (¢173 and ¢175) when using the correspondence typeball.

Upper Case Only Devices

Because these models do not have both uppercase and lowercase characters, the following typing conventions are necessary to enable users to input the full ASCII character set:

1. The keys for letters A through Z input lowercase letters a through z, unless preceded by the escape character \ (left slant). The left slant is shift-L on the keyboard, although it does not show on all keyboards. For example, to input "Smith.ABC", type "\SMITH.\A\B\C".

2. Numbers and punctuation marks map into themselves whenever possible. The underscore (_) is represented by the back arrow (←). The circumflex (^) is represented by the up arrow (↑). The acute accent (´) is represented by the apostrophe (').

3. The following other correspondences exist:

Character	type in	octal
backspace	\-	010
grave accent (`)	\'	140
left brace ({)	\(	173
vertical line (¦)	\!	174
right brace (})	\)	175
tilde (~)	\=	176

Execuport 300

The following non-ASCII graphics are considered to be stylized versions of ASCII characters:

back arrow (←) for underscore (_)

CDI Model 1030

The following non-ASCII graphics are considered to be stylized versions of the ASCII characters:

back arrow (←) for underscore (_)
up arrow (↑) for circumflex (^)

FLOW CONTROL

Some asynchronous terminals implement a flow control protocol for input and/or output. The following paragraphs describe briefly the mechanisms supported by the Multics system.

Input Flow Control

For terminals that can be used to send high-speed input using a paper tape or cassette tape reader, it is useful for the system to be able to instruct the terminal to stop and start transmission so that the input does not arrive faster than it can be processed. Such terminals (for example the Tektronix 4051) suspend transmission on receipt of a particular character (called the input_suspend character), and resume it on receipt of another character (the input_resume character). In addition, such terminals sometimes suspend input at the end of each tape record or block, possibly transmitting the input_suspend character before doing so. It is the responsibility of the system in this case to request the resumption of input by sending the input_resume character. The input_suspend and input_resume characters may be specified in the description of the terminal type as described in Section 3, or by means of the input_flow_control_chars order to the tty_ I/O module, described in Section 6. The timeout option is used to specify that the terminal suspends input without transmitting an input_suspend character, and that the system must send an input_resume character when it detects that input has been suspended. Input flow control is enabled and disabled by means of the iflow mode of the tty_ I/O module.

Output Flow Control

Output flow control is intended to manage terminals that buffer output, since they print or display at less than channel speed. Two types of output flow control protocols are supported by the Multics system. The first, called suspend/resume, is used by various terminals including several made by Digital Equipment Corporation. In this protocol, the terminal sends a particular character (called the output_suspend character) when its buffer is nearly full in order to request that the system temporarily stop sending output. When it is ready to accept more output it sends another character (the output_resume character). The other protocol, called block acknowledgement, is used by various terminals, including the Diablo 1620. In this protocol, the system is expected to subdivide output into blocks no larger than the terminal's buffer, and end each block with a specific character (the end_of_block character). When the terminal is ready to accept more output, it transmits an acknowledgement character. The type of protocol and the specific characters to be used can be specified in the terminal type description as described in Section 3, or by use of the output_flow_control_chars order to the tty_ I/O module, described in Section 6. Output flow control is enabled and disabled by means of the oflow mode of the tty_ I/O module.

BLOCK TRANSFER

Some asynchronous terminals are capable of operating in "block mode", i.e., they can be made to buffer a block of data and then transmit it at channel speed in response to a single keystroke. The system may not handle such high-speed input correctly unless it is informed that the terminal is capable of such transmission. The blk_xfer mode of the tty_ I/O module is used for this purpose.

A terminal is suitable for use in blk_xfer mode if it delimits the block or "frame" of data transferred by appending a specified character (the "frame_end" character) to the block and optionally preceding the block with a "frame_begin" character (which need not be different from the frame_end character). The particular characters used will depend on the terminal. The characters used can be specified by the framing_chars statement in the terminal type definition as described in section 3, or by means of the set_framing_chars order to the tty_ I/O module.

If the terminal is in blk_xfer mode, and frame_begin and frame_end characters have been specified, all characters starting with a frame_begin character, up to and including the next following frame_end character, are treated as a frame. If a frame_end character has been specified, but no frame_begin character has been specified, then all characters between one frame_end character and the next are treated as a frame. In general, none of the characters in a frame are delivered to the user's process until the end of the frame has been reached. Calls to iox_$get_line still read input one line at a time, but the first line in a frame is not available for reading until the entire frame has been received.

SECTION 3

TERMINAL TYPES

TERMINAL TYPE CONCEPT

A terminal type is a named set of parameters identifying the characteristics and behavior of a terminal. The following attributes are components of a terminal type:

- character set
- code set (e.g., EBCDIC, ASCII, etc.)
- behavior in response to carriage movement characters
- behavior in response to other control sequences
- time required for carriage movement functions (delays)
- software control of horizontal tabs
- line length and page length

These parameters are used by MCS to determine how to format output to, and interpret input from, the terminal. The specification of these individual parameters can be changed independently; the terminal type provides a mechanism for specifying them all at once without having to know the details of their implementation.

Terminal Type and Line Type

It is important to distinguish between terminal type and line type, both of which terms are used in describing a terminal connection to Multics. A line type defines the communications protocol used on a particular channel; it is a characteristic of a channel rather than of a terminal. The terminal type may be changed by the user in order to modify the system's treatment of the terminal; the line type is determined by the system, and cannot be changed while the channel is in use.

TERMINAL TYPE TABLE AND TERMINAL TYPE FILE

Terminal types are defined in a data base called the terminal type table (TTT). There is a system-wide TTT that is used by default; each process, however, can use its own TTT instead. The TTT being used by a process can be changed by means of the set_ttt_path command. The various entries of the ttt_info_ subroutine, described in Section 5, can be used to extract information from the TTT. The print_terminal_types command lists the names of all terminal types defined in the TTT; the display_ttt command displays the contents of the TTT in readable format. These commands are all described in Section 4.

The TTT is derived from an ASCII segment, suitable for creation and modification using a text editor, called the terminal type file (TTF). A TTT is generated from a TTF by means of the cv_ttf command, also described in Section 4. The syntax of a TTF is described later in this section.

Setting Terminal Types

Every terminal connected to the Multics system has a terminal type associated with it at all times. The terminal type associated with a particular terminal may be set in any of the following ways:

1. When the terminal dials up (i.e., a connection is established), its terminal type is set in accordance with its line type and baud rate as specified in the default type table in the TTT (see "Syntax of the TTF" below).

2. If the channel on which the terminal dialed up has an initial terminal type associated with it in the channel definition table (CDT), that terminal type is assigned to the terminal. See the MAM Communications for more information on the CDT.

3. If the terminal provides an answerback sequence that matches one of the answerback specifications in the TTT (see "Syntax of the TTF"), its terminal type is set according to the answerback.

4. If the user specifies the -terminal_type control argument to the login command or uses the terminal_type preaccess request, the terminal type is set accordingly. See the description of the login command and terminal_type pre-access request in the MPM Commands.

5. The user may, at any time, change his terminal type by invoking the set_tty command with the -terminal_type control argument.

Changing Terminal Type Definitions

A user wishing to invent a new terminal type, or change the characteristics of an existing terminal type, may edit a copy of the system-supplied TTF and create a new TTT by using the cv_ttf command. Whenever he wishes to use the new or redefined terminal type(s), he switches to the new TTT by means of the set_ttt_path command, and then uses the set_tty command to change his own terminal type to the desired one. This change affects only his current process; other users of the same non-standard TTT are not affected until they use the set_tty command to set or change terminal type.

Note: Various sequences of characters beginning with the ASCII "escape" character (octal 033) are treated by some terminals, when sent as output, as commands to the terminal. These commands may have unexpected or undesirable effects on the behavior of the terminal if, for example, they are embedded in a piece of online mail. For this reason, the standard TTT distributed by Honeywell is designed to prevent the escape character from being included in normal output for most terminal types. Users or sites providing their own TTTs should be aware of the hazards of allowing escape sequences to be sent to terminals as a matter of course.

Terminal Type Table

The terminal type table (TTT), a data base that resides by default in the segment:

```
>system_control_1>ttt
```

describes all the terminal types used by MCS. The initializer requires write access to this segment; all other users require read access.

The TTT is a binary table containing numbers and pointers as well as character strings; therefore, it cannot be examined or modified using the editors. The display_ttt command is used to print out all or part of the TTT; when the system administrator wishes to add or delete terminal types, or change the information about one or more terminal types, he compiles a TTF into a TTT using the cv_ttf command, and then uses the install command to signal the initializer to replace the copy of the TTT in the system.

A TTT is supplied by Honeywell that includes, but is not limited to the following terminal types:

Terminal Type	Description
ASCII_CAPS	Typical ASCII teleprinter terminal (uppercase only)
ASCII_CRT_CAPS	Typical ASCII crt terminal (uppercase only)
ADM3A	Lear Siegler Model ADM-3A
AJ630	Anderson-Jacobson Model 630
AMBASSADOR	Ann Arbor Ambassador CRT
CONCEPT100	Human Designed Systems Concept 100
DIABLO1640	Diablo Systems Series 1640
HAZELTINE1510	Hazeltine Model 1510
HEATH19	Heath Model H19
IBM3271	Control unit for IBM3270 terminal cluster
INFOTON100	Infoton 100 Display Terminal
IRISCOPE200	Iriscope 200
L6FTF	Honeywell L6 File Tranmission Facility
LA120	Digital Equipment LA120 DECwriter III
LED120	Triformation Systems braille terminal
NEC5520	Nippon Electric Model 5520 (Spinwriter)
NEC5525	Nippon Electric Model 5525 (Spinwriter)
SARA	Honeywell SARA 20
SYSTEM75	Selecterm System 75
TEK4023	Tektronix 4023
TEK4025	Tektronix 4025
TELERAY1061	Teleray 1061
TRANSLEX	ECD Translex Intelligent Terminal
TVI920	TeleVideo Model TVI-912 and 920
VIP7700_CLUSTER	Honeywell Multiple Interface Unit for Series VIP7700 Polled VIP Terminal
VIP7705	Honeywell VIP7700 Polled VIP Display Terminal (upper and lower case)
VIP7714	Honeywell VIP7714 read only printer
VIP7760	Honeywell VIP7760 Display Station
VIP7705R	Honeywell VIP7700R Polled VIP Display Terminal (upper and lower case)
VIP7760_CONTROLLER	Honeywell VIP7760 Controller
VIP7804	Honeywell VIP7804 Polled VIP Display Terminal
VIP7804_CLUSTER	Honeywell Multiple Interface Unit for Series VIP7804 Polled VIP Terminals

VISTAR	Infoton Vistar Satellite terminal
VT100	Digital Equipment Model VT-100

These terminal types can change at any time, so the user should invoke the print_terminal_types command to verify the current types.

SYNTAX OF THE TTF

The TTF defines all terminal types known to the system. It is an ASCII file which, when compiled into a binary table (the TTT), is installed by the initializer at the system administrator's request.

The TTF consists of a series of entries describing terminal types, tables, and answerback interpretations. Each entry consists of a series of statements that begin with a keyword and end with a semicolon. White space and comments written in the same style as PL/I comments enclosed by /* and */ may appear between any tokens in the TTF. The last entry in the TTF must be the end statement. Global statements specifying defaults may appear anywhere before the end statement; the defaults they specify are in effect for all subsequent terminal type entries, until they are overridden 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.

Generalized Character Specifications

Many statements in the TTF take as arguments single characters, or lists of single characters. Statements that accept such operands are shown with the <tty_char> notation. A <tty_char> operand may be any of the following:

1. A single unquoted character, such as X, A, p, $ or ~. This notation is only allowed for "simple" characters. This notation may not be used for control characters, white space, ASCII digit characters, "(", ")", "<", ">", "*", ":", ",", ";", or the double quote character.

2. A single quoted character, such as "X", ";", "B", or "0". Any ASCII code can be entered this way. Note that digits should be specified as "0", not 0.

3. A 1 to 3 digit octal number, such as 177, 14 or 007. This enters the character whose octal representation is as specified. Note that 0 is interpreted as octal 000. If the ASCII digit "0" is desired, it must be specified as "0" or 060.

4. The name of a control character, such as DEL. These may be either upper or lower case. All standard control characters are accepted, including:

```
NUL SOH STX ETX EOT ENQ ACK BEL (000 - 007)
BS  TAB LF  VT  FF  CR  SO  SI  (010 - 017)
DLE DC1 DC2 DC3 DC4 NAK SYN ETB (020 - 027)
CAN EM  SUB ESC FS  GS  RS  US  (030 - 037)
```

   In addition, SP (040), DEL (177), NL (012), and HT (011) are also accepted.

5. Control characters may also be entered in the form ^A, which is read as control-A, and is the character sent when the control-A function is used on an ASCII keyboard. ^A is equivalent to SOH, or 001. The letters A-Z (upper or lower case equivalent) preceded by a "^" may be used for 001 through 032. Also accepted are ^@ (000), ^[(033), ^\ (034), ^] (035), ^^ (036), and ^_ (037).

Terminal Type Entry

The entry for each terminal type consists of a terminal_type statement naming the terminal type, followed by various statements describing the attributes of that terminal type. Attributes not specified for a terminal type are set from the defaults established by global statements or supplied by the cv_ttf command.

A description of each statement found in a terminal type entry is given below.

terminal_type: <type name> {like <type name>};

The terminal_type statement is required. It specifies the name of the terminal type described by the statements following it. The type name has a maximum length of 32 characters. All lowercase letters in the type name are translated to uppercase before being stored in the TTT. If the optional like keyword is supplied, it indicates that the attributes of the current terminal type are to be copied from the entry for the type whose name follows the like keyword, except for those that are overridden by subsequent statements in the current entry. The like keyword must refer to a previously defined terminal type.

modes: <mode1>, <mode2>, ... <modeN>;

The modes statement is required. It specifies the modes to be set when the type of the terminal is assigned. A mode name may be preceded by a ^ character to indicate that the specified mode is off for the terminal type. The line-length specification (ll*n*) must be included in the modes statement. For a list of the valid modes, see the description of the tty_ I/O module.

function_keys: <table name>;

The function_keys statement is optional. It specifies the name of a function_key table (defined by a function_key_table entry) to be used for this terminal. If it is omitted, or the table name is a null string, the terminal is assumed to have no function keys.

initial_string: <string>;

The initial_string statement is optional. If present, it specifies a character string to be sent to the terminal in rawo mode in order to initialize certain physical characteristics of the terminal (e.g., to set its horizontal tabs). This string is sent either at dialup time, in response to a "send_initial_string" order, or when set_tty is invoked with the -initial_string control argument. The string is specified as one or more substrings. Each substring may be one of the following:

1. A quoted string; e.g., "sR". If a quoted string is to contain a quote character, that quote must be doubled. (e.g., "s""R" is s"R).

2. <tty-char>

3. (<decimal-integer>) <<substring> ... <substring>>

   where <decimal-integer> is a repetition factor enclosed in parentheses and followed by one or more substrings enclosed in angle brackets (< and>). For example:

   ```
   (10) <040 ETX>
   ```

   represents 10 repetitions of the two character sequence consisting of a space and an ETX character (octal 003).

additional_info: <string>;
The additional_info statement is optional. If provided, it specifies additional information which may be needed to run the terminal. This information is not interpreted by the standard terminal software, and is not passed to the supervisor; it may be used by a special I/O module used to run terminals of the current type. The format and contents of the string depend on the particular application; it may even be the pathname of a segment containing additional information. The string is specified in the same way as for the initial_string statement (above).

bauds: <baud1> <baud2> ... <baudN>;
can also be written as:
bps: <baud1> <baud2> ... <baudN>;
The bauds statement is required if any delay statements (see below) are provided, and it must precede all delay statements. It specifies the baud rates to which the values supplied in the delay statements apply. A specification of "other" in the bauds statement means that the corresponding values in the delay statements apply to all baud rates not specified. If "other" is not specified, then delay values of 0 are assumed for all baud rates not specified in the bauds statement. The following is a list of the baud rates that may be specified:

110	300	1800	7200
133	600	2400	9600
150	1200	4800	19200

cps: <cps1> <cps2> ... <cpsN>;
The cps statement may be used in place of the bauds statement (above) to express terminal speeds in characters per second. The value stored in the TTT is the corresponding baud rate. The cps values that may be specified, and their corresponding baud rates, are listed below:

cps value	baud rate
10	110
15	150
30	300
60	600
120	1200
180	1800
240	2400
480	4800
720	7200
960	9600
1920	19200

Note that there is no way to express a baud rate of 133 in a cps statement.

<delay keyword>: <value1> <value2> ... <valueN>;

In each delay statement, the same number of values must be supplied as baud rates in the bauds, bps, or cps statement. Each value specifies the number of delays to be used for the character described by the delay keyword at the baud rate specified in the corresponding position in the bauds statement (see example below). The possible delay keywords are:

vert_nl_delays
: the number of delays to be sent with a newline operation ($-127 \leq$ vert_nl_delays ≤ 127).

horz_nl_delays
: the variable number of delays to be sent for each column position traversed by a carriage return or a newline operation. This is a floating point number ($0 \leq$ horz_nl_delays ≤ 1).

const_tab_delays
: the minimum number of delays to be sent with a horizontal tab ($0 \leq$ const_tab_delays ≤ 127).

var_tab_delays
: the number of additional delays to be sent for each column position traversed by a horizontal tab. This is a floating point number ($0 \leq$ var_tab_delays ≤ 1).

backspace_delays
: the number of delays to be sent with a backspace ($-127 \leq$ backspace_delays ≤ 127).

vt_ff_delays
: the number of delays to be sent with a vertical tab or formfeed ($0 \leq$ vt_ff_delays ≤ 511).

Negative values for vert_nl_delays and backspace_delays have the same meanings as those described in the description of the set_delay order to the tty_ I/O module. Values of zero are assumed at all baud rates for any delay type not specified.

Example:

bauds:	110	150	300	1200	other;
vert_nl_delays:	2	3	6	24	30;
horz_nl_delays:	.1	.12	.2	.8	1;
const_tab_delays:	0	1	2	7	10;
var_tab_delays:	.1	.12	.2	.8	1;
backspace_delays:	0	0	1	3	6;
vt_ff_delays:	0	0	0	0	0;

The first column gives the complete set of delay values to be used at 110 baud; the second column gives the values to be used at 150 baud, etc.

line_types: <line_type name1>, <line_type name2>, ... <line_type nameN>;

The line_types statement is optional. It specifies the names of the line types on which a terminal of the current type can be run. If it is omitted, the current terminal type can run on any line type.

erase: <tty_char>;

The erase statement is optional. It specifies the erase character for the terminal type. If it is omitted, the # character is used.

kill: <tty_char>;

The kill statement is optional. It specifies the kill character for the terminal type. If it is omitted, the @ character is used.

line_delimiter: <character>;

Specifies the terminal's normal line delimiter character. The character must be specified as one to three octal digits in the terminal's input code (untranslated). This character defaults to 012 unless the line type is 2741 or 1050, in which case it defaults to 055.

keyboard_addressing: yes/no;

The keyboard_addressing statement is optional. It indicates whether or not to do keyboard locking and unlocking for a terminal on a communications channel whose line type is ASCII. If it is not provided, a value of no is assumed. This attribute is ignored for channels of any other line type.

print_preaccess_message: yes/no;

The print_preaccess_message statement is optional. It indicates whether or not the answering service should print a message advising the user to enter a preaccess request if the user entered an unrecognized login word. It is useful in cases where the character code of the terminal may be different from what was expected. At present, only one possible preaccess message is defined, suitable for use with EBCD and Correspondence-code IBM 2741 terminals. If the print_preaccess_message statement is omitted, a value of no is assumed.

conditional_printer_off: yes/no;

The conditional_printer_off statement is optional. It indicates whether or not the answerback identification of the terminal should be used to determine whether the terminal is equipped with the printer-off feature. If yes is specified, a terminal of this type is assumed not to have printer-off unless it has an answerback ID beginning with a digit (0 to 9); otherwise, the existence of the printer-off feature is deduced from the presence or absence of a printer-off sequence in the special characters table (see below). This attribute is primarily useful for IBM 2741 terminals. If the conditional_printer_off statement is omitted, a value of no is assumed.

input_conversion: <table name>;

The input_conversion statement is optional. It specifies the name of a conversion table (defined by a conversion table entry) to be used in converting input from the terminal. If it is omitted, or the table name is a null string, no input conversion table is used.

output_conversion: <table name>;

The output_conversion statement is optional. It specifies the name of a conversion table (defined by a conversion table entry) to be used in converting output sent to the terminal. If it is omitted, or the table name is a null string, no output conversion table is used.

special: <table name>;

The special statement is optional. It specifies the name of a table (defined by a special table entry) to be used as a special characters table when converting input and output (see "Special Characters Table Entry" below). If it is omitted, or the table name is a null string, no special characters table is used. If an output conversion table whose entries are not all 0 is specified, a special characters table must also be specified in order for the terminal to function correctly.

input_translation: <table name>;

The input_translation statement is optional. It specifies the name of a table (defined by a translation table entry) used to translate input from the code of the terminal to ASCII. If it is omitted, or the table name is a null string, input is not translated.

output_translation: <table name>;
The output_translation statement is optional. It specifies the name of a table (defined by a translation table entry) used to translate output from ASCII to the code of the terminal. If it is omitted, or the table name is a null string, output is not translated.

old_type: <number>;
The old_type statement is optional. It may be used for compatibility purposes to specify the numeric value of the terminal type formerly predefined by the Multics Communication System that most closely corresponds to the terminal type described by this terminal type entry.

framing_chars: <frame_begin> <frame_end>;
The framing_chars statement is optional. If present, it specifies the framing characters generated by the terminal when sending frame input at channel speed. The <frame_begin> and <frame_end> are <tty_chars>'s as defined above. In the terminal's character code they represent the frame_begin and frame_end characters respectively (i.e., without translation). <frame_begin> can be NUL or 000 to indicate that there is no frame_begin character; in this case, all input in blk_xfer mode is treated as part of a frame.

The following statements define parameters for flow control to and from asynchronous terminals. For more information, see the discussion of flow control in Section 2.

input_suspend: <tty_char>;
The input_suspend statement is optional. If present, it specifies a character to be transmitted to the terminal in iflow mode in order to temporarily suspend input or, alternatively, a character that the terminal sends to inform the system that it is suspending input. In either case, input is restarted when the input_resume character (see below) is sent to the terminal. This feature is appropriate for use on certain terminals which do input at line speed. If the input_suspend statement is present, the input_resume statement must also be present.

input_resume: <tty_char> {, timeout};
The input_resume statement is optional, unless the input_suspend statement (above) is present. It specifies a character that, when sent to the terminal by the system while in iflow mode, causes it to resume temporarily suspended input. Depending on the terminal, the input_suspend character (above) may not be required. The timeout keyword, if supplied, indicates that the terminal may suspend input (as at the end of a tape record) without transmitting an input_suspend character, in which case it is the responsibility of the system to detect this situation and send the input_resume character after input has been suspended. If the input_resume statement is specified but the input_suspend statement is not, the input_resume statement must include the timeout keyword.

output_suspend: <tty_char>;
The output_suspend statement is optional. It may be used with terminals that implement a suspend_resume protocol for output flow control. If present, it specifies a character that the terminal transmits to cause the system to suspend output so that the terminal can empty its internal buffer. The character is only interpreted by the system in oflow mode. Output is restarted when the terminal sends the output_resume character (see below). If the output_suspend statement is specified, the output_resume statement must also be specified, and none of the output_end_of_block, output_acknowledge, and buffer_size statements may be specified.

output_resume: <tty_char>;
The output_resume statement is optional, unless the output_suspend statement is present. It specifies a character transmitted by the terminal to inform the system that output that was suspended in response to an output_suspend character (see above) can be resumed. If the output_resume statement is present, the output_suspend statement must also be specified, and none of the output_end_of_block, output_acknowledge, and buffer_size statements may be specified.

buffer_size: <number>;
The buffer_size statement is optional. It may be used with terminals that implement a block acknowledgement protocol for output flow control. If present, it specifies the size in characters of the terminal's output buffer, and is used to determine the maximum number of characters to be sent to the terminal at one time (in one transmission) in oflow mode. Each block of up to that number of characters is terminated by an output_end_of_block character (see below). The next block is not transmitted until the terminal sends an output_acknowledge character. If the buffer_size statement is specified, the output_end_of_block and output_acknowledge statements must also be specified, and neither the output_suspend nor the output_resume statement may be specified.

output_end_of_block: <tty_char>;
The output_end_of_block statement is optional. If it is present, it specifies a character to be appended to every output block, as described under the buffer_size statement above. If the output_end_of_block statement is specified, the output_acknowledge and buffer_size statements must also be specified, and neither the output_suspend nor the output_resume statement can be specified.

output_acknowledge: <tty_char>;
The output_acknowledge statement is optional. If present, it specifies a character that is transmitted by the terminal when it is ready to receive the next block of output, as described under the buffer_size statement (above). If the output_acknowledge statement is specified, the buffer_size and output_end_of_block statements must be specified, and neither the output_suspend nor the output_resume statement may be specified.

Video Table Definition

Each terminal type may have an optional video table defined. This table contains control sequences for performing standard operations on video terminals. The table starts with the keyword:

    video_info:

A global video table, which will be used for all terminal types that do not have a video table specified, is started with the keyword:

    Video_info:

The absence of a video table may be specified by:

    video_info: ;

This may be used to negate the effects of a global Video_info statement or a video table inherited from a similar terminal type.

The video_info keyword is followed by 1 or more video info statements, described below. The video table is terminated by the first statement not in this list.

```
screen_height: <decimal-integer>;
```

specifies the usable number of lines on the screen.

```
screen_line_length: <decimal-integer>;
```

specifies the usable number of columns on the screen.

The following statements describe various video control sequences. Each <video_sequence> is a character string built by the concatenation of all the operands given. The sequence may also be followed by an optional delay or padding specification. Video sequences may be built out of any combination of the following:

```
<tty_char>
quoted string, such as "sR"
<addressing|repeat specification>
```

The addressing or repeat specification is entered as follows:

```
({binary|decimal {n}|octal {n}} {X|x|Y|y|N|n} {+|- <tty_char>})
```

This specification takes the value to be sent to the terminal (X,Y,N), encodes it in some way (binary, decimal, octal), and adds or subtracts a fixed offset (+|- <tty-char>).

X represents the horizontal or column position on the screen (0 origin). Y represents the vertical or row position on the screen (also 0 origin). The upper left hand corner of the screen, usually called home, is location X=0, Y=0. The X and Y notations are usually used in the absolute cursor addressing sequence, although they may be wherever required, depending on the terminal. N refers to a repeat count, which some terminals support for some operations.

These values may be encoded in either binary, decimal, or octal. Binary means byte (X), as in the PL/1 builtin. Decimal or octal causes the value to be converted to a character string representation. If {n} is given, it must be 1, 2, or 3, and refers to the length of the character string to be sent, padded with leading zeroes if required. If {n} is 0, or not specified, no leading zeroes will be sent. For example, if X is 35,

```
(decimal 3 X)     ->     "035"
(decimal X)       ->     "35"
(octal 3 X)       ->     "043"
(binary X)        ->     "#"
(X)               ->     "#"
```

If an offset is required, it may be specified as +|- <tty-char>. The value rank (tty-char) will be added to or subtracted from the number to be sent before it is encoded. A common example is (X + SP). In this case, an X of 0 will yield a space (octal 40), an X of 1 will yield "!" (octal 41), etc.

Any video sequence may have an optional <padding> value, expressed as follows:

```
, pad n {us|ms}
```

If us (micro seconds), or ms (milliseconds) is specified, n is interpreted as a time value. Otherwise, it is an absolute number of pad characters required, regardless of the baud rate. If a time is specified, the minimum that can be specified is 100 microseconds. All values are rounded up to the next multiple of 100 microseconds. The maximum value is 26.2 seconds. Time values are converted to a pad count at execution time, depending on the baud rate of the terminal.

The following statements all use the syntaxes just described. Each statement also has a definition of exactly what effect the sequence has on the terminal. If the terminal does not have the capability to perform the function described, the statement should be omitted.

```
abs_pos: <video-sequence> {<padding>} ;
```

defines the absolute cursor positioning sequence. This sequence moves the cursor to a given (X,Y). Other than the cursor, no characters on the screen are affected.

```
clear_screen: <video-sequence> {<padding>} ;
```

defines the screen clearing sequence. This sequence clears the entire screen to spaces regardless of where the cursor is, and leaves the cursor at home. This sequence does not clear tabs.

```
clear_to_eos: <video-sequence> {<padding>} ;
```

defines the clear to end of screen sequence. This clears the screen from the current cursor position to the end of the screen. It does not move the cursor or clear tabs.

```
home: <video-sequence> {<padding>} ;
```

defines the move cursor home sequence. The cursor moves to location X=0, Y=0.

```
clear_to_eol: <video-sequence> {<padding>} ;
```

defines the clear to end of line sequence. Starting at the current cursor position, the rest of the current line clears to spaces. The cursor does not move.

```
cursor_up: <video-sequence> {<padding>} ;
```

defines a sequence to move the cursor up one row. It does not have any effect on the column. The effect of the sequence when the cursor is on the top line of the screen is undefined.

```
cursor_right: <video-sequence> {<padding>} ;
```

defines a sequence to move the cursor one column to the right. It does not have any effect on the row. The effect of the sequence when the cursor is in the last column of the screen is undefined.

cursor_down: <video-sequence> {<padding>} ;

defines a sequence to move the cursor down one row. It does not have any effect on the column. The effect of the sequence when the cursor is on the bottom line of the screen is undefined.

cursor_left: <video-sequence> {<padding>} ;

defines a sequence to move the cursor one column to the left. It does not have any effect on the row. The effect of the sequence when the cursor is in the leftmost column of the screen is undefined.

insert_chars: <video-sequence> {<padding>} ;

defines a sequence for inserting characters on the current line. If end_insert_chars (see next statement) is defined, insert_chars should put the terminal in a mode in which each character sent to the terminal is placed on the screen at the cursor location; each character to the right of the cursor is pushed one position to the right; and the cursor is moved one position to the right. The effect of pushing characters off the righthand edge of the screen is undefined. If end_insert_chars is not defined, insert_chars is defined as opening up N (or 1) spaces on the line, pushing characters to the right of the cursor toward the right. The cursor does not move in this case.

end_insert_chars: <video-sequence> {<padding>} ;

defines a sequence for taking the terminal out of insert_chars mode. See above.

delete_chars: <video_sequence> {<padding>} ;

defines a sequence for deleting characters from the current line. The character at the cursor is deleted, and all characters to the right are moved one column to the left. A space is inserted in the last column of the screen.

insert_lines: <video_sequence> {<padding>} ;

defines a sequence for inserting lines on the screen at the current cursor position. All lines starting at the current line are moved down one line. The current line is filled with spaces. The effect of pushing lines off the bottom of the screen is not defined. This sequence is only defined to work when the cursor is at the leftmost margin. The position of the cursor is not changed.

delete_lines: <video_sequence> {<padding>} ;

defines a sequence for deleting lines from the screen. The current line is deleted by moving all lines below it up one line. The bottom line of the screen is filled with spaces. This sequence is only defined to work when the cursor is at the leftmost margin. The position of the cursor is not changed.

Many terminals do not support all the functions described above, but often they can be simulated by combinations of other functions. For example, the Honeywell VIP7801 does not support clear_screen, as defined, because the clear sequence to that terminal also clears the tabs. The effect of this can be simulated, however, by the combination home (or abs_pos to 0,0) and clear_to_eos, which will clear the screen without affecting the tabs. Thus a clear_screen sequence could be defined which is a concatenation of the other two sequences. Similarly, if a terminal did not have a cursor up sequence, but did support abs_pos, it would be possible to specify a cursor_up sequence as a variant of the abs_pos sequence (by changing the offset by 1). In general, it is <u>not</u> recommended that this sort of optimization be done in the TTF. Instead, the TTF should be viewed as describing the physical characteristics of the terminal, and it is the job of software to choose from among the capabilities of the terminal in order to provide the desired effect.

For most applications, a certain minimal set of functions is required to perform video functions. These are:

1. Some way of clearing the screen. Clear_screen is best, but home and clear_to_eos will work, as well as erase_to_eol on each line.

2. Some way of absolute cursor addressing. Abs_pos is best, but the combination of home and the four cursor motion functions (up, down, left, and right) will work also.

The video_info entry for the Honeywell VIP 7801 is:

```
video_info:
     screen_line_length:  80;
     screen_height:       24;
     home:                ESC H;
     clear_to_eos:        ESC J, pad 1;
     cursor_up:           ESC A;
     cursor_right:        ESC C;
     cursor_down:         LF;
     cursor_left:         BS;
     clear_to_eol:        ESC K;
     insert_chars:        ESC "[I";
     end_insert_chars:    ESC "[J";
     delete_chars:        ESC "[P";
     insert_lines:        ESC "[L";
     delete_lines:        ESC "[M";
     abs_pos:             ESC f (X + " ")(Y + " ");
```

Global Statements

A global statement specifies a default value for a terminal type attribute. It has the same form as the statement describing the attribute in a terminal type entry, except that the statement keyword begins with a capital letter. Global statements may not appear within terminal type entries. Global statements may be used for any of the statements listed above for a terminal type entry, except for terminal_type, initial_string, additional_info, and the delay statements. (A global Bauds, Bps, or Cps statement is allowed, although a global delay statement is not.) A global video table definition may be given by using the statement:

```
     Video_info:
```

followed by one or more video table entries. The statement:

```
     Video_info: ;
```

may be used to specify that no default video table exists.

Conversion Table Entry

A conversion table entry consists of two statements: one specifying the name of the table and one specifying its contents. The following is a description of a conversion table entry.

conversion_table: <table name>;
<value0> <value1> ... <value255>;

The table name is a string of up to 32 characters. The values are octal numbers of one to three digits; each value is the indicator corresponding to the character whose ASCII value is the index of the indicator in the table. See the descriptions of the set_input_conversion and set_output_conversion orders to the tty_ I/O module for a description of conversion tables and the indicators they contain. If fewer than 256 values are supplied, the unspecified values are assumed to be zero.

Translation Table Entry

A translation table entry consists of a statement specifying the name of the table and a statement specifying its contents, as described below.

translation_table: <table name>;
<value0> <value1> ... <value255>;

The table name is a string of up to 32 characters. The values are octal numbers of one to three digits. Each value is the result of translation of the character whose bit representation is the index into the table of that value (i.e., <value0> is the result of translating a character represented as 000, <value8> corresponds to a character represented as 010, etc.). If fewer than 256 values are supplied, the unspecified values are assumed to be zero.

Function Key Table Entry

A function key table is begun and named by a function_key_table statement, which is the only required statement. All the remaining statements define function key sequences, and are optional. A function key is defined by giving the name of the key, and the characters transmitted when the key is struck. The following names are recognized: home, up, down, left, right, and key(_i_), where _i_ must be 0 or greater, and is the number of the function key. If the terminal has no function key labelled 0, then the first key may be 1. No gaps are permitted, but the keys may be defined in any order.

Up to four sequences may be defined for each key, giving the sequences transmitted for the function key, the function key when shifted, the function key when the control key is held down, and the function key with both shift and control, in that order, separated by commas, and terminated by a semi-colon. If less than four sequences are given, or a sequence is missing, the terminal is assumed to not have a function key for that combination of key-strokes.

If the terminal always takes some local action (e.g. clearing the screen, moving the cursor) (possibly in addition to transmitting the sequence) when a key is struck, it is better to omit the sequence entirely, since most applications will not want the side-effect to occur, and would most likely not even use the key.

Example

```
function_key_table: vip_7801_function_keys;
     home: ESC H;
     left: ESC D;
     right: ESC C;
     up: ESC A;
     down: ESC B;
     key(0): ESC e, ESC `, ESC c;
     key (1): ESC 0, ESC 1;
     key (2): ESC 2, ESC 5;
     key (3): ESC 6, ESC 7;
     key (4): ESC 8, ESC 9;
     key (5): ESC :, ESC ";";
     key (6): ESC <, ESC =;
     key (7): ESC >, ESC ?;
     key (8): ESC P, ESC Q;
     key (9): ESC R, ESC S;
     key (10): ESC T, ESC V;
     key (11): ESC \, ESC ];
     key (12): ESC ^, ESC _;
```

Special Characters Table Entry

A special characters table entry consists of a special_table statement and a set of statements specifying the contents of a special characters table. These statements are described below. Wherever the expression <sequence> appears, it means from zero to three <tty_char>s, separated by white space, representing a sequence of characters to be output to fulfill the specified function. If any statement specifying a sequence is omitted, a null sequence is assumed, unless otherwise specified in the description of the statement. All sequences are in ASCII code except for the printer_on and printer_off sequences. For those sequences that are used when specific indicators are encountered in the output conversion table, the relevant indicator is given in the description of the statement. See the description of the various tables in the discussion of orders to the tty_ I/O module for more detailed information.

special_table: <table name>;
     The special_table statement specifies the name of the table. It is a string of up to 32 characters.

new_line: <sequence>;
     The new_line statement specifies the sequence to be output for a newline character (output conversion indicator 1).

carriage_return: <sequence>;
     The carriage_return statement specifies the sequence to be output for a carriage return character (output conversion indicator 2). If the sequence is null, backspaces are used to move the carriage to the left margin.

backspace: <sequence>;

The backspace statement specifies the sequence to be output for a backspace character (output conversion indicator 4). If the sequence is null, a carriage return and spaces are used to reach the correct column. The carriage return and backspace sequences should not both be null.

tab: <sequence>;

The tab statement specifies the sequence to be output for a horizontal tab character. If the sequence is null, an appropriate number of spaces is used to reach the next tab stop.

vertical_tab: <sequence>;

The vertical_tab statement specifies the sequence to be output for a vertical tab character (output conversion indicator 5) if the terminal is in vertsp mode.

form_feed: <sequence>;

The form_feed statement specifies the sequence to be output for a formfeed character (output conversion indicator 6) if the terminal is in vertsp mode.

printer_on: <sequence>;

The printer_on statement specifies the sequence to be output to fulfill a "printer_on" order. The sequence is specified in the character code of the terminal. If the sequence is null, the printer_on feature is not supported.

printer_off: <sequence>;

The printer_off statement specifies the sequence to be output to fulfill a "printer_off" order. The sequence is specified in the character code of the terminal. If the sequence is null, the printer_off feature is not supported.

red_shift: <sequence>;

The red_shift statement specifies the sequence to be output for a red-ribbon-shift character (output conversion indicator 10 (octal)).

black_shift: <sequence>;

The black_shift statement specifies the sequence to be output for a black-ribbon-shift character (output conversion indicator 11 (octal)).

end_of_page: <sequence>;

The end_of_page statement specifies the sequence to be output when output is suspended because the page length of the terminal has been reached. If it is omitted, the character sequence "EOP" is assumed. A null string indicates that output is to stop at the right margin of the last line of a page.

output_escapes: <indicator1> <sequence1>,
<indicator2> <sequence2>, ... <indicatorN> <sequenceN>;

The output_escapes statement specifies the escape sequences to be output for characters whose output conversion indicators are 21 (octal) or greater when the terminal is in ^edited mode. The indicators specified in the statement are the same as the corresponding indicators in the output conversion table.

edited_output_escapes: <indicator1> <sequence 1>,
<indicator2> <sequence2>, ... <indicatorN> <sequenceN>;

The edited_output_escapes statement specifies sequences like those specified by the output_escapes statement, but they are used when the terminal is in edited mode.

input_escapes: <value1> <result1>,
<value2> <result2>, ... <valueN> <resultN>;

The input_escapes statement specifies those input characters that are to be interpreted as escape sequences when preceded by an escape character, and the resulting characters that replace those sequences. (An escape character in this context is a character defined by software to initiate an escape sequence, i.e., one with an indicator of 2 in the input conversion table.) Each "value" is an octal number representing the ASCII value of a character that is used in an escape sequence; the corresponding "result" is an octal number representing the single character that replaces the escape sequence in the input stream.

Default Types

Exactly one default_types statement must appear in the TTF. It specifies default terminal types on the basis of baud rate and line type. When a terminal dials up, this information is used by the answering service to assign its type if no default terminal type is specified in the CDT entry for the channel. The default_types statement is described below.

default_types: <baud1> <line_type1> <terminal_type1>,
<baud2> <line_type2> <terminal_type2>, ...
<baudN> <line_typeN> <terminal_typeN>;

Each baud*i* is a number representing a baud rate, or the word "any"; each line_type*i* is the name of a valid line type, or the word "any"; each terminal_type*i* is the default terminal type for the specified combination of baud rate and line type. The table thus constructed is searched in the order in which the baud rate, line_type, terminal_type triplets are specified, and the first entry that matches the particular channel is used to determine the initial terminal type. The last entry in the table should specify "any" for both baud rate and line type.

Answerback Table

The answerback table consists of entries specifying how to determine a terminal type and identification on the basis of its answerback. The answerback sent by the terminal is scanned under control of each answerback table entry, starting with the first one specified in the answerback table. If the scan succeeds (as described below), and the line type of the terminal is one that is valid for the terminal type specified in the answerback table entry, the terminal type and ID are derived from that entry; otherwise, the answerback is rescanned using the next entry, and so on. An answerback table entry consists of two statements: an answerback statement and a type statement.

answerback: <keyword1> <value1>, <keyword2> <value2>, ... <keywordN> <valueN>;

The answerback statement describes how the scan of the answerback is to be performed. The "scan pointer," indicating the current character position in the answerback of the scan, starts at the beginning of the answerback string and is adjusted according to the controls specified by the answerback statement. The possible keyword-value pairs are described below.

match <expression>

<expression> is either the word "digit," the word "letter," or a string enclosed in quotes. If it is digit or letter, the scan fails unless the character addressed by the scan pointer is a digit (0 to 9) or a letter (A to Z or a to z), respectively. If it is a quoted string, the scan fails unless the scan pointer points to the beginning of a matching string. If the match succeeds, the scan pointer is advanced over the matching string or character, and the scan is continued using the next keyword-value pair.

search <expression>
: works like match, except that the scan succeeds if the matching character or string is found anywhere to the right of the scan pointer.

skip N
: causes the scan pointer to be moved N characters to the right. The value N may be negative, in which case the pointer is actually moved to the left. The scan fails if there are fewer than N characters between the scan pointer and the end (or beginning if N is negative) of the answerback string.

id N
: the N characters starting at the right of the scan pointer form the ID of the terminal. The value N must be in the range 1 <= N <= 4. If there are fewer than N characters to the right of the scan pointer, the scan fails.

id rest
: as many characters (up to 4) as remain to the right of the scan pointer constitute the ID of the terminal (not including control and carriage-motion characters).

type: <type name>;
: The type statement specifies the name of the terminal type to be assigned to a terminal whose answerback satisfies the specification in the answerback statement. The specified terminal type must be defined by a previous terminal type entry. If the type statement is omitted, the answerback is to be used to set the ID only, and the terminal type is not changed.

Preaccess Commands

The preaccess command entries are used to define the terminal types to be set in response to preaccess commands at dialup time. Each preaccess command entry consists of a preaccess_command statement and a type statement. See the MPM Commands for more information about preaccess commands.

preaccess_command: <command>;
: The preaccess_command statement specifies the name of a preaccess command. The three commands currently supported are MAP, 963, and 029. If a preaccess command statement is not present for any one of these command statements, the command statement has no effect when entered from the terminal.

type: <type name>;
: The type statement specifies the terminal type to be assigned when the corresponding command is entered. The specified type must be defined by a previous terminal type entry.

Examples

```
/* Sample terminal type entries */


Input_conversion:  standard_input_conv;

terminal_type:  1050;
 modes:  default,hndlquit,tabs,red,ll130;
 bauds:                    133;
 vert_nl_delays:             1;
 horz_nl_delays:           .11;
 const_tab_delays:           1;
 var_tab_delays             .2;
 input_translation:  ebcdic_input_trans;
 output_translation:  ebcdic_output_trans;
 output_conversion:  ebcdic_output_conv;
 special:  ebcdic_special;
 line_types:  1050;
 old_type:  1;


terminal_type:  2741 like 1050;
 modes:  default,hndlquit,tabs,red,ll125;
 conditional_printer_off:  yes;
 print_preaccess_message:  yes;
 line_types:  2741;
 old_type:  2;


terminal_type:  TN300;
 modes:  default,hndlquit,tabs,ll118;
 initial_string:  ESC "2" CR ESC "1" (11) < (10) (SP) ESC "1";
 bauds:                  110   150   300  1200;
 vert_nl_delays:           0     2     6   -38;
 backspace_delays:        -2    -3    -6   -27;
 vt_ff_delays:            19    29    59   230;
 output_conversion:  ascii_output_conv;
 special:  tn300_special;
 line_types:  ASCII, 202ETX;
 old_type:  4;


/* sample default_types statement and answerback entries */

default_types:
                  110   ASCII  TTY33,
                  any   ASCII  ASCII,
                  any     VIP  ASCII,
                  133    1050   1050,
                  133    2741   2741,
                 1200    ARDS   ARDS,
                 1200  202ETX  TN300,
                  any     any   G115;
```

```
/* the match below sets the terminal type to 1050 if the line type is 1050 */

answerback:         id 1;
 type:              1050;

answerback:         search "0", id 3;
 type:              2741;

answerback:         search "0";
 type:              2741;

answerback:         search " E", id 3;
 type:              TN300;

answerback:         search " E";
 type:              TN300;


/* sample conversion, translation, and special tables */

conversion_table:  standard_input_conv;
          03 00 00 00 00 00 00 00
          00 00 01 00 04 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 05 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 02 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 00
          00 00 00 00 00 00 00 03;


translation_table:  ebcdic_input_trans;
          040 055 100 046 070 161 171 150
          064 155 165 144 000 000 000 000
          062 153 163 142 060 000 000 000
          066 157 167 146 000 010 000 000
          061 152 057 141 071 162 172 151
          065 156 166 145 000 012 000 011
          063 154 164 143 043 044 054 056
          067 160 170 147 000 000 000 000
          040 137 134 053 052 121 131 110
          072 115 125 104 000 000 000 000
          074 113 123 102 051 000 000 000
          047 117 127 106 000 010 000 000
          075 112 077 101 050 122 132 111
          045 116 126 105 000 012 000 011
          073 114 124 103 042 041 174 136
          076 120 130 107 000 000 000 000;

special_table:  ebcdic_special;
 new_line:  012;
 carriage_return:  ;
 backspace:  10;
 tab:  11;
 vertical_tab: ;
 form_feed:  ;
 printer_on:  15;
 printer_off:  16;

 red_shift:  033 141;
 black_shift:  033 142;
 end_of_page:  105 117 120;

 output_escapes:
     21 134 074,              /* esc < ([) */
     22 134 076,              /* esc > (]) */
     23 134 047,              /* esc . (^) */
     24 134 050,              /* esc ( ({) */
     25 134 051,              /* esc ) (}) */
     26 134 164;              /* esc t (~) */

 edited_output_escapes:
     21 050 010 075,          /*  (= ({) */
     22 051 010 075,          /*  )= (}) */
     23 047,                  /*  .  (~) */
     24 050 010 055,          /*  (- ({) */
     25 051 010 055,          /*  )- (}) */
     26 047 010 136;          /*  .~ (~) */

 Input_escapes:
     074 133,                 /* esc < -> [ */
     076 135,                 /* esc > -> ] */
     047 140,                 /* esc . -> ~ */
     050 173,                 /* esc ( -> { */
     051 175,                 /* esc ) -> } */
     164 176,                 /* esc t -> ~ */
     124 176;                 /* esc T -> ~ */

end;
```

SECTION 4

COMMANDS

This section contains descriptions of commands used for communications I/O.

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 3 of the MPM Commands.

Name: cv_ttf

The cv_ttf command compiles a terminal type file (TTF) into a terminal type table (TTT), in preparation for installing it.

Usage

```
cv_ttf path {-control_arg}
```

where:

1. path
   is the pathname of the TTF to be compiled. The TTT that results from the compilation is placed in the user's working directory; its entryname is the same as the entryname of the TTF with the suffix ttt added.

2. control_arg
   may be either of the following:

   -long, -lg
   specifies that all error messages produced by cv_ttf are to be printed in long form.

   -brief, -bf
   specifies that all error messages produced by cv_ttf are to be printed in short form.

Notes

If neither -long nor -brief is specified, the first instance of a given error produces a long message, and all subsequent instances of that error produce short messages.

Name: dial_out

The dial_out command enables a user to access a remote system by dialing a specified destination over a dial-out channel (i.e., a channel that has been configured with the autocall service type, as specified in the service statement of the CMF).

Usage

```
dial_out channel {destination} {-control_args}
```

where:

1. channel
   the name of the dial-out channel to be used. The star convention is allowed, which means the answering service selects a channel that has a matching name and matching attributes (if specified).

2. destination
   is the dial-out destination (e.g., phone number or network address) to be used in making the connection. If this argument is omitted, a channel is attached as described under dial_manager_$privileged_attach in the MPM Subsystem Writers' Guide.

   This argument can be up to 32 characters in length and can include dial-tone-wait characters, which suspend dialing until the autocall unit receives a dial tone. The standard FNP multiplexer recognizes the exclamation point ("!") as the dial-tone-wait character and pauses at each one encountered to await a dial tone.

3. control_args
   may be chosen from the following:

-raw
suppresses Multics terminal management and makes the dial_out interface completely transparent. Characters are transmitted directly to or from the foreign system, without any conversion or processing.

-echo
causes characters entered by the user to be echoed locally.

-line
causes the communications line to transmit line-at-a-time rather than character-at-a-time. This permits the dialing of an FNP channel that cannot be run at line speed in character mode.

-escape STR, -esc STR
sets the escape character to STR. The escape character enables the user to enter dial_out requests from within the dial_out environment. The default escape character is the exclamation point ("!").

-terminal_type STR, -ttp STR
sets the terminal type of the remote connection to STR. This argument is useful in those cases where the host has unusual communications requirements.

-resource STR, rsc STR
: specifies the desired characteristics of the dial-out channel. STR (which can be null) consists of reservation attributes separated by commas. The channel used by a dial_out operation must have the characteristics specified in the reservation string. Reservation attributes consist of a keyword and optional argument. Attributes allowed are:

```
baud_rate=BAUD_RATE
line_type=LINE_TYPE
```

where BAUD_RATE is a decimal representation of the desired channel line speed and LINE_TYPE is a valid line type, chosen from line_types.incl.pl1 (see set_line_type, in Section 6).

-abbrev
: enables the user to invoke abbrev processing of request lines.

-profile PATH
: defines the pathname PATH of the profile segment that contains abbrevs used with the dial_out command. The suffix .profile is assumed if it is not present. The default is the user's current profile segment.

-request STR
: Causes the dial_out command to execute the request STR after the connection is established, but before entering the dial_out environment.

Notes

The user may enter dial_out requests from within the dial_out environment by preceding requests with the escape character ("!" by default). Typed entries between one escape character and the next or the end of the line are interpreted as the dial_out request. The escape character itself may be sent by entering it twice in succession.

Use of the dial_out command requires the dialok attribute and rw access to >sc1>rcp>NAME.acs.

List of Requests

escape STR, esc STR
: sets the escape character to STR.

file_output PATH, fo PATH
: copies output to a file identified by the pathname PATH.

interrupt, int, break, brk, ip
: sends an interrupt signal (line break) to the foreign system.

modes {raw|^raw},{echo|^echo}
: allows the user to enable or disable the raw and echo modes (as described under the -raw and -echo control arguments).

revert_output, ro
reverts the effect of the previous file_output request; i.e., causes output to no longer be copied to the file identified by the file_output request.

send
causes arguments within the request to be sent to the foreign system as if they were typed by the user.

send_file PATH, sf PATH
causes the contents of pathname PATH to be sent to the foreign system.

switch_name
returns the name of the I/O switch used by the dial_out interface.

Examples

The dial_out command

```
dial_out b.h218 9-555-5622 -raw
Ready on tty_ b.h218 -destination 9-555-5622
```

attaches channel b.h218 to the dial destination 9-555-5622, while suppressing Multics terminal management. The ready message is printed by the system, verifying the connection.

The dial_out command

```
dial_out b.h000.1.* 31060849
```

establishes a network connection over an appropriate channel (i.e., one that meets the star convention requirements) at address 31060849.

This page intentionally left blank.

Name: display_ttt

The display_ttt command prints all or part of a terminal type table (TTT) on the user's terminal, or outputs it to a file. The format of the output is such that it can be used as a terminal type file (TTF).

Usage

```
display_ttt {-control_args}
```

where control_args may be chosen from the following list:

-pathname path, -pn path
: specifies that the TTT whose pathname is path is to be displayed. If this control argument is omitted, the process' current TTT is displayed.

-terminal_type name, -ttp name
: specifies that only the terminal type entry for the terminal type named name is to be displayed (see "Notes" below).

-table name, -tb name
: specifies that only the conversion, translation, function keys, or special table named name is to be displayed (see "Notes" below).

-output_file path, -of path
: specifies that output is to be directed to the file whose pathname is path. If this control argument is omitted, output is directed to the terminal.

-header, -he
: specifies that a header is to be printed (see "Notes" below).

-no_header, -nhe
: specifies that no header is to be printed (see "Notes" below).

Notes

If neither -terminal_type nor -table is specified, the entire contents of the TTT are displayed; if -no_header also is not specified, an introductory comment is printed, giving the pathname of the TTT, the date, and the user_id of the author of the original TTT. If either -terminal_type or -table is specified, only the specified terminal type entry or table is displayed, without the introductory comment unless -header is also specified.

Name: l6_ftf

The l6_ftf command allows a process to handle file transfer requests from a Level 6, using the L6 File Transfer Facility (FTF) protocol (referred to as L6 TRAN; see Level 6/Level 6 File Transmission Facility User's Guide CB33). This command continues to listen for and carry out Level 6 requests until the user explicitly tells it to stop. Only sequential ASCII or sequential binary files may be transferred to or from the Level 6. ASCII files on Multics are assumed to be stream files when sending, and are stored as stream files when receiving. Binary files on Multics have a special format (see Notes below).

Usage

```
     l6_ftf channel_name {control_args}
```

where:

1. channel_name
   is the name of a polled VIP subchannel over which the file transfers will take place. It must have the "x" prefix. (See Notes below).

2. control_arg
   may be either of the following:

   -long, -lg
   prints a line describing each file transfer as it starts and as it is completed. The default is not to print this information.

   -target_dir PATH, -td PATH
   specifies that the pathnames of any files to be transferred are relative to the target directory. The root may be specified as ">", which allows absolute pathnames to be specified. The default is the working directory.

Notes

This command continues to listen for and process file transfer requests from the Level 6 on the specified channel until the Multics user types "q" or "quit" or the channel disconnects. The quit request may be typed at any time, but will only take effect before any file transfer has started or between two file transfers.

The user must have rw access to the ACS of the specified channel name to use the file transfer facility. The user must have the "dialok" attribute turned on in the PDT. The polled VIP subchannel must have the slave attribute in the CDT, and must be an "X" type subchannel (see the description of polled VIP multiplexers in MAM Communications.)

Interrupting and releasing a file transfer in the middle of the transfer may result in aborting the operation in an inconsistent state, and causing the Level 6 task to hang.

The Polled VIP multiplexer must have a terminal type (in the TTF) which sets the "additional_info" parameter to "max_message_len=1009 omit_nl=yes omit_ff=yes". (See the description of polled VIP multiplexers in MAM Communications).

Only sequential ASCII or sequential binary files may be transferred from or created on the Level 6. On Multics, ASCII files are assumed to be or are created as stream files. Notice that blank lines in a Level 6 file actually have some character on them, usually a space or tab. These characters will end up in the Multics file. The command sends blank lines from Multics files to the Level 6 by sending a line containing a single space character.

On Multics, binary files are sequential vfiles. Each record is assumed to have the following format:

```
     dcl 1 binary_record aligned based,
            2 num_sextets fixed bin(35) aligned,
            2 sextets (0 refer binary_record.num_sextets) fixed bin(6)
               unsigned unaligned;
```

Each binary record is stored in a vfile_ record of size currentsize(binary_record) * 4.

Examples

The following Level 6 command:

```
TRAN ISD -L6 -N >SPD>PVE01 -ISA TEST -ASA FOO
```

sends the Level 6 file TEST to the Multics segment FOO assuming the 16_ftf command has specified the PVE subchannel name corresponding to >SPD>PVE01 on the Level 6. See CB33 for more information (this is a Level 6 manual).

Name: print_terminal_types, ptt

The print_terminal_types command prints the names of all terminal types defined in the terminal type table (TTT) currently in use. If the TTT in use is not the system default TTT, the command prints the current TTT's pathname at the head of the list of terminal names.

Usage

```
     print_terminal_types {path}
```

where path specifies the pathname of the TTT. If omitted, the current TTT is used.

Name: print_ttt_path

This command prints the name of the terminal type table (TTT) segment currently in use. This is the pathname last set by a set_ttt_path command, or the pathname of the default system TTT.

Usage

```
print_ttt_path
```

No arguments are required.

Name: set_ttt_path

The set_ttt_path command changes the pathname of the terminal type table (TTT) associated with the user's process.

Usage

```
set_ttt_path {path} {-control_arg}
```

where:

1. path
   is the pathname of the TTT. If no path argument is given, then control_arg is required.

2. control_arg
   can be -reset (-rs) to reset the TTT pathname to its default value of >system_control_1>ttt.

Notes

The use of path argument and the -reset control argument are mutually exclusive; only one may be given in any invocation of the set_ttt_path command.

Name: set_tty, stty

The set_tty command modifies the terminal type associated with the user's terminal and/or various parameters associated with terminal I/O. The type as specified by this command determines character conversion and delay timings; it has no effect on communications line control.

Usage

```
set_tty {-control_args}
```

where control_args may be chosen from the following control arguments:

-all, -a
: is the equivalent of specifying the four control arguments -print, -print_edit, -print_frame, and -print_delay.

-buffer_size N, -bsize N
: specifies the terminal's buffer size to be used for output block acknowledgement (see the discussion of output flow control in Section 2). N is the terminal's buffer size in characters. If the end_of_block and acknowledgement characters have not been specified (either as part of the terminal type description or by means of the -output_etb_ack control argument to set_tty), this control argument may not be specified.

-brief, -bf
: may only be used with the -print control argument and causes only those modes that are on plus those that are not on/off type modes (e.g., ll79) to be printed.

-delay STR, -dly STR
: sets the delay timings for the terminal according to STR, which is either the word "default" or a string of six decimal values separated by commas. If "default" is specified, the default values for the current terminal type and baud rate are used. The values specify vert_nl, horz_nl, const_tab, var_tab, backspace, and vt_ff, in that order. The meanings of the values are as follows:

vert_nl
: is the number of delay characters to be output for all newlines to allow for the linefeed ($-127 \leq$ vert_nl ≤ 127). If it is negative, its absolute value is the minimum number of characters that must be transmitted between two linefeeds (for a device such as a TermiNet 1200).

horz_nl
: is a number to be multiplied by the column position to obtain the number of delays to be added for the carriage return portion of a newline ($0 \leq$ horz_nl ≤ 1). The formula for calculating the number of delay characters to be output following a newline is:

```
ndelays = vert_nl + fixed (horz_nl*column)
```

const_tab
: is the constant portion of the number of delays associated with any horizontal tab character ($0 \leq$ const_tab ≤ 127).

var_tab
: is the number of additional delays associated with a horizontal tab for each column traversed ($0 \leq$ var_tab ≤ 1). The formula for calculating the number of delays to be output following a horizontal tab is:

```
ndelays = const_tab + fixed (var_tab*n_columns)
```

backspace
: is the number of delays to be output following a backspace character ($-127 \leq$ backspace ≤ 127). If it is negative, its absolute value is the number of delays to be output with the first backspace of a series only (or a single backspace). This is for terminals such as the TermiNet 300 that need delays to allow for hammer recovery in case of overstrikes, but do not require delays for the carriage motion associated with the backspace itself.

vt_ff
: is the number of delays to be output following a vertical tab or formfeed ($0 \leq$ vt_ff ≤ 511).

The horz_nl and var_tab values are floating-point numbers; all other values are integers. If any of the six values is omitted, the corresponding delay value is not changed; if values are omitted from the end of the list, trailing commas are not required.

-edit edit_chars, -ed edit_chars
: changes the input editing characters to those specified by edit_chars. The edit_chars control argument is a 2-character string consisting of the erase character and the kill character, in that order. If the erase character is specified as a blank, the erase character is not changed; if the kill character is omitted or specified as a blank, the kill character is not changed.

-frame STR, -fr STR
: changes the framing characters used in blk_xfer mode to those specified by STR, where STR is a 2-character string consisting of the frame-begin and the frame-end character, respectively. These characters must be specified in the character code of the terminal, and may be entered as octal escapes, if necessary. The frame-begin character is specified as a NUL character to indicate that there is no frame-begin character; the same is true for a frame-end character. These characters have no effect unless blk_xfer mode is on. It is an error to set the frame-end character to NUL if the frame-begin character is not also set to NUL.

-initial_string, -istr
: transmits the initial string defined for the terminal type to the terminal.

-input_flow_control STR, -ifc STR
sets the input_suspend and input_resume characters to those specified in STR, which is a string of one or two characters. (See the discussion of input flow control in Section 2.) If STR contains two characters, the first character is the input_suspend character and the second one is the input_resume character. If STR contains only one character, it is the input_resume character and there is no input_suspend character.

-io_switch STR, -is STR
specifies that the command be applied to the I/O switch whose name is STR. If this control argument is omitted, the user_i/o switch is assumed.

-modes STR, -md STR
sets the modes for terminal I/O according to STR, which is a string of mode names, each separated by a single comma. Many modes can be optionally preceded by "^" to turn the specified mode off. For a list of valid mode names, see the description of the tty_ I/O module. Modes not specified in STR are left unchanged. See "Notes" below.

-output_etb_ack STR, -oea STR
sets the output_end_of_block and output_acknowledge characters to those specified in STR, which is a string of two characters. (See the discussion of output flow control in Section 2.) The first character of STR is the end_of_block character and the second one is the acknowledge character. If a buffer size has not been specified (either as part of the terminal type description or by means of the -buffer_size control argument to set_tty), this control argument may not be specified.

-output_suspend_resume STR, -osr STR
sets the output_suspend and output_resume characters to those specified in STR, which is a string of two characters. (See the discussion of output flow control in Section 2.) The first character of STR is the output_suspend character and the second is the output_resume character.

-print, -pr
prints the terminal type and modes on the terminal. If any other control arguments are specified, the type and modes printed reflect the result of the command.

-print_delay, -pr_dly
prints the delay timings for the terminal.

-print_edit, -pr_ed
prints the input-editing characters for the terminal.

-print_frame, -pr_fr
prints the framing characters for the terminal.

-reset, -rs
: sets the modes to the default modes string for the current terminal type.

-terminal_type STR, -ttp STR
: sets the terminal type of the user to STR, where STR can be any one of the types defined in the terminal type table (TTT). The default modes for the new terminal type are turned on and the initial string for the terminal type, if any, is transmitted to the terminal. Refer to the print_terminal_types command for information on obtaining a list of terminal types currently in the TTT.

Notes

Invoking the set_tty command causes the system to perform the following steps in the specified order:

1. If the -terminal_type control argument is specified, set the specified type, turn on the default modes for that type and send the initial string for that type.
2. If the -reset control argument is specified, set the modes to the default modes string for the current terminal type.
3. If the -modes control argument is specified, turn on or off those modes explicitly specified.
4. If the -initial_string control argument is specified, transmit the initial string to the terminal.
5. If the -edit control argument is specified, set the editing characters.
6. If the -frame control argument is specified, set the framing characters.
7. If the -delay control argument is specified, set the delay values.
8. If the -input_flow_control control argument is specified, set the input flow control characters.
9. If the -buffer_size, -output_etb_ack, or -output_suspend_resume control argument is specified, set the corresponding output flow control parameters.
10. If the -print control argument is specified, print the type and modes on the terminal.
11. If the -print_edit control argument is specified, print the editing characters on the terminal.
12. If the -print_frame control argument is specified, print the framing characters on the terminal.
13. If the -print_delay control argument is specified, print the delay values on the terminal.

Examples

The command line:

```
set_tty -delay 6,0,0,0,-6,59
```

sets all six delay values to those used by a TermiNet 300.

The command line:

```
set_tty -delay 5,0.6,,,2,63
```

sets the delay values so that 5 delays will be output with a newline, plus 3 more for every 5 columns of carriage return; 2 delays will be used for each backspace, 63 for a vertical tab or formfeed, and whatever values were already in force for horizontal tabs.

The command line:

```
set_tty -delay ,1.3,,.8
```

sets horz_nl to 1.3 and var_tab to 0.8, while leaving all other delay values as they were before.

The command line:

```
set_tty -frame \002\003
```

sets the frame-begin and frame-end characters to the ASCII STX and ETX characters, respectively.

SECTION 5

SUBROUTINES

This section describes the ttt_info_ subroutine, which extracts information about a terminal type from the terminal type table.

The conventions shown in the usage lines of this subroutine are the same as those used in the MPM Subroutines; briefly, the usage lines first show the proper format to use when declaring the subroutine, and then show a sample call. For a complete description of all the usage line conventions, refer to Section 2 of the MPM Subroutines.

Name: ttt_info_

The ttt_info_ subroutine extracts information from the terminal type table (TTT).

Entry: ttt_info_$terminal_data

This entry point returns a collection of information that describes a specified terminal type.

Usage

```
declare ttt_info_$terminal_data entry (char(*), fixed bin, fixed bin, ptr,
     fixed bin(35));

call ttt_info_$terminal_data (tt_name, line_type, baud, ttd_ptr, code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. line_type (Input)
   is a line type number against which the compatibility of the terminal type is verified. If nonpositive, the line type number is ignored. For further description, see the tty_ I/O module in Section 6.

3. baud (Input)
   is a baud rate used to select the appropriate delay table.

4. ttd_ptr (Input)
   is a pointer to a structure in which information is returned. (See "Notes" below.)

5. code (Output)
   is a standard status code. If the terminal type is incompatible with the line type, a value of error_table_$incompatible_term_type is returned.

Notes

The ttd_ptr argument should point to the following structure (terminal_type_data.incl.pl1):

```
dcl 1 terminal_type_data           aligned,
    2 version                      fixed bin,
    2 old_type                     fixed bin,
    2 name                         char(32) unaligned,
    2 tables,
          3 input_tr_ptr           ptr,
          3 output_tr_ptr          ptr,
          3 input_cv_ptr           ptr,
          3 output_cv_ptr          ptr,
          3 special_ptr            ptr,
          3 delay_ptr              ptr,
         2 editing_chars          unaligned,
          3 erase char(1)          unaligned,
          3 kill char(1)           unaligned,
        2 framing_chars            unaligned,
          3 frame_begin            char(1) unaligned,
          3 frame_end              char(1) unaligned,
        2 flags,                   unaligned,
          3 keyboard_locking       bit(1),
          3 input_timeout          bit(1),
          3 output_block_acknowledge  bit(1),
          3 mbz                    bit(15),
        2 line_delimiter           char(1) unaligned,
        2 mbz                      bit(9) unaligned,
        2 flow_control_chars       unaligned,
          3 input_suspend          char(1),
          3 input_resume           char(1),
          3 output_suspend_etb     char(1),
          3 output_resume_ack      char(1),
        2 output_buffer_size       fixed bin;
```

where:

1. version (Input)
   is the version number of the above structure. It must be 1 or 2.

2. old_type (Output)
   is the old terminal type number that corresponds to the terminal type name. (The old terminal type number is provided only for compatibility with the obsolete tty_ order requests set_type and info.) A value of -1 indicates that no corresponding old type exists.

3. name (Output)
   is the terminal type name.

4. input_tr_ptr (Output)
   is a pointer to a structure containing the input translation table. This structure is identical to the info structure for the set_input_translation order of the tty_ I/O module.

5. output_tr_ptr (Output)
   is a pointer to a structure containing the output translation table. This structure is identical to the info structure for the set_output_translation order of the tty_ I/O module.

6. input_cv_ptr (Output)
   is a pointer to a structure containing the input conversion table. This structure is identical to the info structure for the set_input_conversion order of the tty_ I/O module.

7. output_cv_ptr (Output)
   is a pointer to a structure containing the output conversion table. This structure is identical to the info structure for the set_output_conversion order of the tty_ I/O module.

8. special_ptr (Output)
   is a pointer to a structure containing the special characters table. This structure is identical to the info structure for the set_special order of the tty_ I/O module.

9. delay_ptr (Output)
   is a pointer to a structure containing the delay table. This structure is identical to the info structure for the set_delay order of the tty_ I/O module.

10. erase (Output)
   is the erase character.

11. kill (Output)
   is the kill character.

12. frame_begin (Output)
   is the frame-begin character.

13. frame_end (Output)
   is the frame-end character.

14. keyboard_locking (Output)
   indicates whether the terminal type requires keyboard locking and unlocking.

"1"b	yes
"0"b	no

15. input_timeout (Output)
   is "1"b if the timeout option was specified on an input_resume statement in the TTF.

16. output_block_acknowledge (Output)
   is "1"b if output_end_of_block and output_acknowledge statements were specified in the TTF.

17. mbz
   must be "0"b.

18. line_delimiter (Output)
   is the line delimiter character.

19. flow_control_chars
   identifies the flow control characters. It is not present if version (above) is 1.

20. input_suspend (Output)
   is the character sent to the terminal to suspend input, or sent by the terminal to indicate that it is suspending input.

21. input_resume (Output)
   is the character sent to the terminal to resume input.

22. output_suspend_etb (Output)
   is the character sent by the terminal to suspend output if output_block_acknowledge is "0"b; otherwise it is the character to be appended to each output block.

23. output_resume_ack (Output)
    is the character sent by the terminal to resume output if output_block_acknowledge is "0"b; otherwise it is the character used to acknowledge an output block.

24. output_buffer_size (Output)
    is the size, in characters, of the terminal's buffer, for use with a block acknowledgement protocol. It is 0 unless output_block_acknowledge is "1"b. It is not present if version is 1.

Entry: ttt_info_$modes

This entry point returns the default modes for a specified terminal type.

Usage

```
declare ttt_info_$modes entry (char(*), char(*), fixed bin(35));

call ttt_info_$modes (tt_name, modes, code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. modes (Output)
   is the default modes string for the terminal type. If its length is less than 256 characters, the entire modes string is not necessarily returned.

3. code (Output)
   is a standard status code.

Entry: ttt_info_$preaccess_type

This entry point returns the terminal type name associated with a specified preaccess request.

Usage

```
     declare ttt_info_$preaccess_type entry (char(*), char(*), fixed bin(35));

     call ttt_info_$preaccess_type (request, tt_name, code));
```

where:

1. request (Input)
   is one of the following three preaccess requests: MAP, 963, or 029.

2. tt_name (Output)
   is the name of the associated terminal type. Its length should be at least 32 characters.

3. code (Output)
   is a standard status code.

Entry: ttt_info_$additional_info

This entry point returns additional information for a specified terminal type to be used by I/O modules other than tty_.

Usage

```
     dcl ttt_info_$additional_info entry (char(*), char(*) varying,
          fixed bin(35));

     call ttt_info_$additional_info (tt_name, add_info, code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. add_info (Output)
   is the additional information string. If no additional information is defined for the terminal type, a null string is returned. Maximum length is 512 characters.

3. code (Output)
   is a standard status code.

Entry: ttt_info_$initial_string

This entry point returns a string that can be used to initialize terminals of a specified terminal type. The string must be transmitted to the terminal in raw output (rawo) mode. The initial string is most commonly used to set tabs on terminals that support tabs set by software.

Usage

```
declare ttt_info_$initial_string entry (char(*), char(*) varying,
     fixed bin(35));

call ttt_info_$initial_string (tt_name, istr_info, code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. istr_info (Output)
   is the initial string. If no initial string is defined for the terminal type, a null string is returned. Maximum length is 512 characters.

3. code (Output)
   is a standard status code.

Entry: ttt_info_$dialup_flags

This entry point returns the values of two flags for a specified terminal type.

Usage

```
declare ttt_info_$dialup_flags entry (char(*), bit(1), bit(1),
     fixed bin(35));

call ttt_info_$dialup_flags (tt_name, ppm_flag, cpo_flag, code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. ppm_flag (Output)
   indicates whether a preaccess message should be printed when an unrecognizable login line is received from a terminal of the specified type:

"1"b	yes
"0"b	no

3. cpo_flag (Output)
   indicates whether "conditional printer off" is defined for the terminal type, i.e., if the answerback indicates whether a terminal is equipped with the printer off feature:

"1"b	yes
"0"b	no

4. code (Output)
   is a standard status code.

Entry: ttt_info_$decode_answerback

This entry point decodes a specified answerback string into a terminal type name and terminal identifier.

Usage

```
     declare ttt_info_$decode_answerback entry (char(*), fixed bin, char(*),
          char(*), fixed bin(35));

     call ttt_info_$decode_answerback (ansb, line_type, tt_name, id, code);
```

where:

1. ansb (Input)
   is the answerback string.

2. line_type (Input)
   is a line type number with which the decoded terminal type must be compatible. A nonpositive line type number is ignored. For further description, see the tty_ I/O module.

3. tt_name (Output)
   is the terminal type name decoded from the answerback. Its length should be at least 32 characters. If no terminal type is indicated, a null string is returned.

4. id (Output)
   is the terminal identifier decoded from the answerback. Its length should be at least four characters. If no id is indicated, a null string is returned.

5. code (Output)
   is a standard status code.

Entry: ttt_info_$encode_type

This entry point obtains a code number that corresponds to a specified terminal type name.

Usage

```
declare ttt_info_$encode_type entry (char(*), fixed bin, fixed bin(35));

call ttt_info_$encode_type (tt_name, type_code, code);
```

where:

1. tt_name (Input)
   is the terminal type name.
2. type_code (Output)
   is the corresponding terminal type code number.
3. code (Output)
   is a standard status code.

Entry: ttt_info_$decode_type

This entry point obtains the terminal type name that corresponds to a specified terminal type code number.

Usage

```
declare ttt_info_$decode_type entry (fixed bin, char(*), fixed bin(35));

call ttt_info_$decode_type (type_code, tt_name, code);
```

where:

1. type_code (Input)
   is the terminal type code number.
2. tt_name (Output)
   is the corresponding terminal type name.
3. code (Output)
   is a standard status code.

Entry: ttt_info_$video_info

This entry point is used to obtain a copy of the video sequences table for a particular terminal type.

Usage

```
     dcl  ttt_info_$video_info entry (char (*), fixed bin, ptr, ptr, fixed bin
          (35));

     call ttt_info_$video_info (terminal_type, baud_rate, areap, ttyvtblp,
          rode);
```

where:

1. terminal_type (Input)
   is the name of the terminal type for which the video table is required.

2. baud_rate (Input)
   is the current baud rate of the terminal. This may be set to 0 if it is unknown, or uninteresting.

3. area (Input)
   is a pointer to an area where the video table may be allocated. If null, the system free area is used.

4. ttyvtblp (Output)
   is a pointer to the video table, if present.

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

The format of a video table is given in the include file tty_video_tables.incl.pl1.

```
     dcl 1 tty_video_table       aligned based (ttyvtblp),
         2 version               fixed bin,
         2 screen_height         fixed bin,
         2 screen_line_length    fixed bin,
         2 scroll_count          fixed bin,
         2 flags                 unaligned,
           3 overstrike_available bit (1) unal,
           3 automatic_crlf      bit (1) unal,
           3 simulate_eol        bit (1) unal,
           3 pad                 bit (33) unaligned,
         2 video_chars_len       fixed binary (21)
         2 pad                   (2) bin (36)
         2 nseq                  fixed bin,
         2 sequences             (N_VIDEO_SEQUENCES refer (tty_video_table.nseq)
                                  like tty_video_seq aligned,
         2 video_chars           char (tty_video_table_video_chars_len refer
                                  (tty_video_table.video_chars_len)) unal;
```

where:

1. version
   is the version of this structure. It must be tty_video_tables_tables_version_1, also declared in this include file.

2. screen_height
   is the number of lines on this terminal.

3. screen_line_length
   is the number of character positions (columns) in each line.

4. scroll_count
   is the number of lines scrolled upward when a scroll command is sent to the terminal (if the terminal is capable of scrolling). For most terminals this will be 1. A value of 0 indicates that one line is scrolled.

5. flags
   describe characteristics of the terminal.

6. overstrike_available
   is "1"b if the terminal can overstrike (i.e., more than one character can be seen in the same character position).

7. automatic_crlf
   is "1"b if the terminal performs a carriage return and line feed when a character is displayed in the last column.

8. pad
   has an undefined value, and is reserved for future expansion

8. simulate_eol
   is reserved for future expansion.

9. pad1
   is reserved for future expansion.

10. video_chars_len
    specifies the length of the string containing all video sequences.

11. pad
    is reserved for future expansion.

12. nseq
    is the number of the highest video sequence defined for this terminal. Not all sequences are defined for all terminals, so programs should check this value before indexing the sequence array.

13. sequences
    is an array of video sequences. Each element of the array specifies the character sequence for a video control operation. The indices for specific sequences are defined by constants also declared in this include file. See below.

14. video_chars
    is a string holding concatenations of all video sequences.

The include file defines values for the indices into the array of sequences for the video operations supported. The names of these values are: ABS_POS, CLEAR_SCREEN, CLEAR_TO_EOS, HOME, CLEAR_TO_EOL, CURSOR_UP, CURSOR_RIGHT, CURSOR_DOWN, CURSOR_LEFT, INSERT_CHARS, END_INSERT_CHARS, DELETE_CHARS, INSERT_LINES, DELETE_LINES. The include file also defines N_VIDEO_SEQUENCES, which is the number of the highest index ever defined.

A video sequence is defined by the structure tty_video_seq, defined in the include file tty_video_tables.incl.pl1.

```
dcl 1 tty_video_seq      based (ttyvseqp) aligned,
      2 flags            unaligned,
        3 present        bit (1) unal,
        3 interpret      bit (1) unal,
        3 able_to_repeat bit (1) unal,
        3 cpad_present   bit (1) unal,
        3 cpad_in_chars  bit (1) unal,
        3 pad            bit (7) unaligned,
        3 general        bit (6) unaligned,
      2 cpad             fixed bin (18) unsigned unaligned,
      2 pad              bit (15) unal,
      2 len              fixed bin (9) unsigned unaligned,
      2 seq_index        fixed bin (12) unsigned unaligned;
```

where:

1. present
   is "1"b if the operation is supported.

2. interpret
   is "1"b if the sequence contains the encoding of the line, column, or repeat count and must be inspected more closely.

3. able_to_repeat
   is "1"b if the terminal can perform multiple sequences of this operation by receiving a single character sequence containing the repeat count; the repeat count is encoded in the sequence.

4. cpad_present
   is "1"b if the terminal requires padding after the operation.

5. cpad_in_chars
   is "1"b if the padding is in characters, or "0"b if the padding is in tenths of milliseconds. If the baud rate is supplied to the ttt_info_$video_info subroutine, then padding will always be expressed in characters.

6. pad
   is reserved for future expansion.

7. general
   is reserved for future expansion to define per-sequence information.

8. cpad
   is the padding count in units defined by cpad_in_chars.

9. pad
   is reserved for future expansion.

10. len
    is the length of the string of characters defining this sequence.

11. seq_index
    is the index of the start of the string in tty_video_table.video_chars.

Many terminals allow a repetition count to be supplied with an operation (e.g., to delete multiple lines). Positioning operations require line and column coordinates. These values must be expressed in some encoding. A variety of encodings are supported. Parameters to be transmitted are specified by an encoding character in the video sequence string. An encoding character is a nine bit byte whose high order bit is set and is defined by the structure tty_numeric_encoding in the include file tty_video_tables.incl.pl1. The encoding scheme is described in the write up for the video_info table of the Terminal Type file.

```
dcl 1 tty_numeric_encoding based unaligned,
      2 flags,
        3 must_be_on        bit (1) unal,
        3 express_in_decimal bit (1) unal,
        3 express_in_octal  bit (1) unal,
        3 offset_is_0       bit (1) unal,
      2 l_c_or_n            fixed bin (2) unsigned unaligned,
      2 num_digits          fixed bin (2) unsigned unaligned,
      2 pad                 bit (1) unaligned
      2 offset              fixed bin (8) unaligned;
```

where:

1. must_be_on
   is "1"b for an encoding character.

2. express_in_decimal
   is "1"b if the value should be expressed as decimal digits.

3. express_in_octal
   is "1"b if the value should be expressed in octal digits. If both flags are off, the value should be sent as a single character.

4. offset_is_0
   if "0"b, the following byte is a fixed bin(8) value to be added to the value before encoding. If "1"b, the offset is 0 and the next byte has no special significance.

5. l_c_or_n
   specifies the type of value to be encoded. Its value may be 0, 1, or 2, and indicates that this encoding character specifies the line number, column number, or repeat count respectively.

6. num_digits
   specifies the number of digits to be sent. A value of 0 causes all significant digits to be sent, with leading zeroes suppressed.

7. pad
   is reserved for future expansion.

8. offset
   is present only if offset_is_0 is "0"b. It gives an offset to be added to the value before expressing it in octal or decimal.

Entry: ttt_info_$function_key_data

This entry point returns a collection of information describing the function keys of a specified terminal type.

Usage:

```
dcl ttt_info_$function_key_data entry (char(*), ptr, ptr, fixed bin (35));

call ttt_info_$function_key_data (tt_name, areap, function_key_data_ptr,
     code);
```

where:

1. tt_name (Input)
   is the terminal type name.

2. areap (Input)
   points to an area where the function_key_data info structure may be allocated. If null, the system free area is used. If the area is not large enough, the area condition is signalled.

3. function_key_data_ptr (Output)
   points to the function_key_data structure allocated by this entry point. The structure is described below.

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

Notes

The data structure allocated by this routine is declared in the include file function_key_data.incl.pl1.

```
     dcl 1 function_key_data aligned based (function_key_data_ptr),
           2 version fixed bin,
           2 highest fixed bin,
           2 sequence,
             3 seq_ptr pointer,
             3 seq_len fixed bin (21),
           2 cursor_motion_keys,
             3 home (0:3) like key_info,
             3 left (0:3) like key_info,
             3 up (0:3) like key_info,
             3 right (0:3) like key_info,
             3 down (0:3) like key_info,
           2 function_keys (0:function_key_data_highest refer
               (function_key_data.highest), 0:3) like key_info;

     dcl (KEY_PLAIN init (0),
          KEY_SHIFT init (1),
          KEY_CTRL init (2),
          KEY_CTRL_AND_SHIFT init (3)
          ) fixed bin internal static options (constant);

     dcl 1 key_info unaligned based (key_info_ptr),
           2 sequence_index fixed bin (12) unsigned unaligned,
           2 sequence_length fixed bin (6) unsigned unaligned;
```

where:

1. version
   is the version of this structure. It should be set to function_key_data_version_1.

3. highest
   is the number of the highest function key defined.

3. sequence
   defines the character string holding the concatenation of all the sequences. The sequence for a given key is defined as a substring of this string.

4. seq_ptr
   is the address of the string.

5. seq_len
   is its length.

6. cursor_motion_keys
   defines some miscellaneous keys whose names connote motion of the cursor. Note that the meaning of these keys is defined only by the application, which may or may not choose to take advantage of mnemonic value of these key legends.

7. home
   defines the sequences for the HOME key, used by itself, with SHIFT, with CONTROL, and with SHIFT and CONTROL. An absent sequence will have a sequence length of zero.

8. left
   defines the left arrow key in the same way as HOME is defined.

9. up
   defines the up arrow key.

10. right
    defines the right arrow key.

11. down
    defines the down arrow key.

12. function_keys
    defines the sequences for the function keys of the terminal. If the terminal has no function key labelled "0", all sequences for 0 will have zero length.

13. key_info
    defines a given sequence.

14. sequence_index
    is the index of the beginning of the sequence in the string of all sequences.

15. sequence_length
    is the length of the sequence. If the length is zero, the sequence is not present.

Mnemonic values are defined for the subscripts for various key combinations: KEY_PLAIN, KEY_SHIFT, KEY_CTRL, and KEY_CTRL_AND_SHIFT.

For example, the sequence for the LEFT arrow key with SHIFT would be:

```
substr (function_key_seqs,
     function_key_data.left(KEY_SHIFT).sequence_offset,
     function_key_data.left(KEY_SHIFT).sequence_length)
```

SECTION 6

INPUT/OUTPUT MODULES

This section describes the tty_ I/O module, as well as the special purpose communications I/O modules. The conventions used in giving the formats of the attach descriptions are the same as those for the usage lines of commands.

Name: bisync_

The bisync_ I/O module performs stream I/O over a binary synchronous communications channel.

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

Attach Description

```
bisync_ device {-control_args}
```

where:

1. device
   is the name of the communications channel to be used for communications (see Appendix A for a discussion of channel names).

2. control_args
   can be chosen from the following:

   -size N
   sets to N the number of characters to be transmitted in each bisync block. The default is 256 characters.

   -ascii
   uses the ASCII bisync protocol. This is the default.

   -ebcdic
   uses the EBCDIC bisync protocol.

   -transparent
   uses the transparent bisync protocol. This is the default.

   -nontransparent
   uses the nontransparent bisync protocol.

   -bretb
   causes the get_chars operation to return any block of data ending with an end of text block (ETB) character. The default is to return only blocks ending with an end of text (ETX) control character or an intermediate text block (ITB) control character (see the discussion of the get_chars operation below).

   -breot
   causes the get_chars operation to return any block of data ending with an end of transmission (EOT) character (see the discussion of the get_chars operation below).

   -hangup
   causes an automatic hangup when the switch is detached.

   -bid_limit N
   sets to N the number of times a line bid is retried. The default is 30 times.

-ttd_time N
: sets to N the number of seconds of temporary text delay (TTD) transmissions if output is delayed. The default is 2 seconds.

-ttd_limit N
: sets to N the maximum number of TTDs that are sent before sending an EOT. The default is 30 TTDs.

-multi_record {N}
: specifies that blocking of logical records is done by the I/O module. If specified, N is the maximum number of records per block. If N is not given, the number of records per block is as many as fit.

Open Operation

The bisync_ I/O module supports the stream_input, stream_output, and stream_input_output opening modes.

Put Chars Operation

The put_chars entry splits the data to be written into blocks according to the -size control argument in the attach description. The appropriate bisync control characters are added to the beginning and end of each block. Each block except the last is transmitted with an ETB control character at the end. The last block is transmitted with an ETX control character at the end.

Get Chars Operation

The get_chars entry reads and decodes bisync blocks, removes the control characters, and returns the message text to the caller's buffer.

Characters are returned up to the next logical bisync break character. Normally this is ETX. If -bretb is specified in the attach description, ETB is also considered to be a break character. If -multi_record is specified, the inter-record ITB characters are also considered to be break characters. In addition, if -breot is specified, error_table_$end_of_info is returned when an EOT is read.

Get Line Operation

The get_line entry reads and decodes bisync blocks, removes the control characters, and returns the message text to the caller's buffer. Characters are returned until either a newline character is placed in the buffer, or the buffer is filled. The get_line entry does not distinguish between blocks ending in ETB or ITB and blocks ending in ETX.

Control Operation

Several of the control operations supported by the bisync_ I/O module are identical to those supported by the tty_ I/O module, and are documented there. They include:

abort
resetread
resetwrite
hangup
read_status
write_status
event_info

The following additional control operations are supported by this I/O module.

set_bid_limit
: where info_ptr points to a fixed binary bid limit to replace the bid_limit specified in the attach description.

get_bid_limit
: where info_ptr points to a fixed binary bid limit that is set either to the value specified at attach or in the last get_bid_limit order.

set_bsc_modes
: where info_ptr points to a structure of the following form:

```
dcl 1  bsc_modes,
    2  transparent bit(1) unal,
    2  ebcdic bit(1) unal,
    2  mbz bit (34) unal;
```

The setting of the transparent and ebcdic bits then replaces the values specified in the attach description.

get_bsc_modes
: returns the structure described under set_bsc_modes.

runout
: has meaning only in multi-record mode and writes the current partially filled block.

set_size
: where info_ptr points to a fixed binary buffer size. This new size replaces the size specified in the attach description. It may not be larger than the size originally specified in the attach description.

get_size
: where info_ptr points to a fixed binary buffer size and returns the current size.

set_multi_record_mode
    where info_ptr points to a fixed binary record count. If the count is 1, the I/O module enters single record mode. Otherwise, multi-record mode is entered and the count specifies the maximum number of records per block. Zero (or a null info_ptr) specifies no fixed limit; i.e., as many records as fit are blocked.

get_multi_record_mode
    where info_ptr points to a fixed binary record count. This order returns the multirecord record count. A 1 indicates single record mode.

send_nontransparent_msg
    writes the data specified in nontransparent bisync mode, regardless of the current transparency mode. This order is used to send short nontransparent control sequences while in transparent mode. The info_ptr points to a structure of the following form:

```
dcl 1  order_msg,
    2  data_len fixed bin,
    2  data char (order_msg.data_len);
```

end_write_mode
    causes the I/O module to block until all outstanding output has been written.

get_chars
    performs a get_chars operation and returns additional information about the input. The info_ptr points to a structure of the following form:

```
dcl 1  get_chars_info,
    2  buf_ptr ptr,
    2  buf_len fixed bin(21),
    2  data_len fixed bin(21),
    2  hbuf_ptr ptr,
    2  hbuf_len fixed bin(21),
    2  header_len fixed bin(21),
    2  flags,
       3  etx bit(1) unal,
       3  etb bit(1) unal,
       3  soh bit(1) unal,
       3  eot bit(1) unal,
       3  pad bit(32) unal;
```

where:

buf_ptr, buf_len (Input)
    define an input buffer for the text of the message.

data_len (Output)
    is set to the number of characters of text read.

hbuf_ptr, hbuf_len (Input)
    define an input buffer for the header of the message.

header_len (Output)
    is set to the header's length in characters.

etx (Output)
    indicates that text is terminated with an etx character.

etb (Output)
indicates that text is terminated with an etb character.

soh (Output)
indicates that the data includes a header.

eot (Output)
indicates that an eot was received.

pad (Output)
is unused space in this structure.

hangup_proc
sets up a specified event call channel to be signalled over, and a procedure to be called, if the communications channel hangs up. The hangup_proc input structure has the following form:

```
dcl 1 hangup_proc  aligned,
      2 entry      entry variable,
      2 datap      ptr,
      2 prior      fixed bin;
```

where:

entry
is the entry to call when a hangup is detected.

datap
is a pointer to data for the hangup procedure.

prior
is the ipc_ event call priority to be associated with hangup notification.

Modes Operation

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

Name: g115_

The g115_ I/O module performs stream I/O to a remote I/O terminal that has the characteristics of the Honeywell Level 6 remote batch facility (G115 type). The hardware options currently supported are defined by the control arguments described below.

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

Attach Description

g115_ -control_args

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

-device STR
: attaches the subdevice specified by STR. STR may be printer, punch, reader, or teleprinter.

-auto_call N
: specifies the phone number, N, to be called via the auto call unit on the specified communications channel.

-tty STR
: connects the remote I/O terminal to the communications channel named STR.

-comm STR
: uses the communications I/O module specified by STR. Currently, the only permissible value for STR is "rci". This argument is required for compatibility with all other I/O modules used by the I/O daemon.

-ascii
: uses the ASCII character set. This is the default. This argument is accepted for compatibility with other terminal I/O modules.

-physical_line_length N, -pll N
: specifies the physical line length, N, of the output device. This argument is accepted for compatibility with other terminal I/O modules.

-terminal_type STR, -ttp STR
: STR specifies the terminal type whose conversion, translation, and special tables defined in the user or system terminal type table (TTT) are used to convert and translate input and output to and from the device. If not specified, no conversion or translation is performed. For more information about the allowable conversion values see "Notes" below.

Open Operation

The g115_ I/O module supports stream_input, stream_output, and stream_input_output opening modes.

Put Chars Operation

The put_chars entry blocks the data to be written into blocks of up to 324 characters and transmits them to the specified communications channel.

Get Chars Operation

The get_chars entry reads blocks of up to 324 characters and returns the number of characters requested up to the next record separator.

Control Operation

This I/O module supports all the control operations supported by the tty_ I/O module. In addition, it supports the following:

select_device
: selects the subdevice, either printer, punch, or teleprinter, to which output is next directed. The input structure is of the form:

```
dcl device char(32);
```

runout
: transmits any data stored in the output buffer. There is no input structure.

hangup_proc
: sets up a specified event call channel to be signalled over, and a procedure to be called, if the communications channel hangs up. The hangup_proc structure has the following form:

```
dcl 1 hangup_proc aligned,
      2 entry entry variable,
      2 datap ptr,
      2 prior fixed bin;
```

where:

entry
: is the entry to call when a hangup is detected.

datap
: is a pointer to data for the hangup procedure.

prior
: is the ipc_ event call priority to be associated with hangup notification.

reset
: sets the edited mode of output conversion.

end_write_mode
: prevents the g115_ module from returning until all outstanding output has been written to the attached channel.

Modes Operation

This I/O module supports the rawi and rawo modes. It also supports the nonedited and default modes, which set and reset the edited output conversion, if it has been enabled by the -terminal_type control argument.

Notes

The only allowable values in the output conversion table are 00 and any values greater than 16. All values defined in the description of the tty_ I/O module are allowed for input conversion. Input and output translation tables may be up to 256 characters in length.

Name: hasp_host_

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

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

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

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

Attach Description

```
hasp_host_ -control_args
```

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

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

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

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

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

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

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

Open Operation

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

Write Record Operation

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

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

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

where:

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

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

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

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

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

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

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

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

Read Record Operation

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

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

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

where:

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

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

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

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

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

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

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

element_size (Output)
is always set to 9.

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

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

Control Operation

This I/O module supports the following control operations:

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

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

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

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

where:

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

input_available (Output)
    indicates whether input is available:

```
"0"b   no input
"1"b   input
```

resetread
    discards any pending input.

resetwrite
    discards any as yet unprocessed output.

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

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

where:

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

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

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

A detailed explanation of data_ptr and priority may be found in the description of ipc_ in the MPM Subsystem Writer's Guide.

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

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

Modes Operation

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

Character Set Specification

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

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

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

SIGNON Processing

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

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

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

signon_record CONTROL ORDER

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

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

where:

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

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

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

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

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

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

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

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

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

no_signon_record CONTROL ORDER

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

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

Notes

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

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

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

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

Name: hasp_workstation_

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

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

This module must be attached to a subchannel of a communications channel configured to use the HASP ring-0 multiplexer. (See the description of the HASP multiplexer in MAM Communications.)

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

Attach Description

```
hasp_workstation_ -control_args
```

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

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

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

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

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

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

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

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

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

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

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

Open Operation

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

Write Record Operation

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

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

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

where:

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

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

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

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

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

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

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

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

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

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

Read Record Operation

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

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

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

where:

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

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

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

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

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

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

element_size (Output)
is always set to 9.

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

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

Control Operation

This I/O module supports the following control operations:

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

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

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

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

where:

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

input_available (Output)
indicates whether input is available:

"0"b	no input
"1"b	input

resetread
flushes any pending input.

resetwrite
flushes any as yet unprocessed output.

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

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

where:

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

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

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

A detailed explanation of data_ptr and priority may be found in the description of ipc_ in the MPM Subsystem Writer's Guide.

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

Modes Operation

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

Character Set Specification

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

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

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

Carriage Control Specifications

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

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

The format of this carriage control specification is:

```
Tn:Tn:Tn:...
```

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

For example, the string:

```
c7:s5:c12
```

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

Name: ibm2780_

The ibm2780_ I/O module performs stream I/O to a remote I/O terminal that has the characteristics of an IBM 2780 data transmission terminal. The hardware options currently supported are defined by the control arguments described below.

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

This module in turn constructs an attach description for the module specified in the -comm control argument, passing the attach information for ascii or ebcdic, tty, transparent or nontransparent, and all other attach information specified by the caller.

Attach Description

```
ibm2780_ -control_args
```

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

-ascii
: transmits control information and data in ASCII.

-ebcdic
: converts control information and data to its EBCDIC representation before transmission. This is the default.

-multi_record
: transmits multiple records (up to 7) as a block, rather than separately. The default is single record transmission.

-physical_line_length N, pll N
: sets the maximum character width of the remote I/O terminal printer to N characters. The default is 80 characters. This variable is used to set tabs and pad records if the transparent option is specified.

-horizontal_tab, -htab
: supports tab control on the remote I/O terminal printer. Tabs are set every 10 spaces. The default is no tab control.

-tty STR
: connects the remote I/O station to the communications channel named STR.

-comm STR
: uses the communications I/O module specified by STR.

-transparent
: uses a transparent communication protocol.

-nontransparent
: uses a nontransparent communication protocol. This is the default.

-device STR
: specifies that this attachment is associated with the device STR. Currently, it is accepted only for compatibility with other I/O modules.

-carriage_ctl STR
: the eight-character string STR, taken two characters at a time, sets the four carriage control characters that specify the advance of 0, 1, 2, and 3 lines. The default set of characters is ESC/, ESC/, ESCS, and ESCT, where the mnemonic ESC means the ASCII escape character.

-slew_ctl STR
: the six-character string STR, taken two characters at a time, sets the slew control characters that specify top of form, inside page, and outside page. The default set of characters is ESCA, ESCA, and ESCA.

-printer_select STR
: the two-character string STR sets the printer select. The default printer select string is ESC/.

-punch_select STR
: the two-character string STR sets the punch select. The default punch select string is ESC4.

-terminal_type STR, -ttp STR
: STR specifies the terminal type whose conversion, translation, and special tables defined in the user or system terminal type table (TTT) are used to convert and translate input and output to and from the device. If not specified, no conversion or translation is performed. For more information about the allowable conversion values see "Notes" below.

Open Operator

The ibm2780_ I/O module supports stream_input, stream_output, and stream_input_output opening modes.

Put Chars Operation

The put_chars entry splits the data to be written into blocks of 80 or 400 characters, depending on whether multirecord mode is enabled, and transmits the number of characters specified to the specified communications I/O module. The blocks are of fixed or variable length, depending on whether transparent mode is enabled or not, respectively.

Get Chars Operation

The get_chars entry reads characters up to 80 or 400 characters, depending on whether multirecord is enabled, and returns the number requested, up to the next record separator.

Control Operation

This I/O module supports all the control operations supported by the communications I/O module specified in the attach description. In addition, it supports the following:

set_bsc_modes
: sets the character mode, either ascii or ebcdic and transparency. The input structure is defined as follows:

```
dcl 1 set_bsc_modes aligned,
      2 char_mode bit(1), unaligned,
      2 transparent bit(1) unaligned;
```

where char_mode = "1"b if ebcdic and "0"b if ascii, and transparent = "1"b if transparency is enabled and "0"b if not.

select_device
: selects the subdevice (either printer, punch or teleprinter) to which output is next directed. The input structure is of the form:

```
dcl device char(32) based;
```

set_multi_record_mode
: sets the number of records per block. The input structure is of the form:

```
dcl record_number fixed bin based;
```

Modes Operation

This module supports the nonedited and default modes, which set and reset the edited output conversion, if it has been enabled by the -terminal_type control argument.

Notes

The only allowable values in the output conversion table are 00 and any values greater than 16. All values defined in the description of the tty_ I/O module are allowed for input conversion. Input and output translation tables may be up to 256 characters in length.

Name: ibm3270_

The ibm3270_ I/O module performs stream I/O to and from an IBM 3270 Information Display System (or any compatible device) over a binary synchronous communications channel.

NOTE: Do not use this module to communicate with a 3270 device over a multiplexed channel. Use the tty_ module in that case.

This module description assumes a knowledge of the IBM 3270 communications protocol as described in the IBM 3270 Information Display System Component Description, Order No. GA27-2749-4.

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

Attach Description

```
ibm3270_ device {-control_args}
```

where:

1. device
   is the name of the communications channel to be used.

2. control_args
   can be chosen from the following:

   -async
   specifies that the I/O module is to return to its caller immediately after performing a read order (described below under "Control Operation") when input is not available, rather than blocking and waiting for a response from the device.

   -ebcdic
   uses the EBCDIC bisync protocol and character code. This is the default.

   -ascii
   uses the ASCII bisync protocol and character code.

Open Description

This I/O module supports only the stream_input_output opening mode. If the -async control argument is specified in the attach description (see above), the open operation may return the status code error_table_$request_pending; in this case, the caller should perform an event_info order (see below, "Control Operation") and block on the returned event channel; when the process receives a wakeup on this channel, the open operation should be retried.

Put Chars Operation

This I/O module does not support the put_chars operation. Output is sent to the device by means of the write order (see "Control Operation" below).

Get Chars Operation

This I/O module does not support the get_chars operation. Input is read from the device by means of the read order (see "Control Operation" below).

Get Line Operation

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

Control Operation

This I/O module supports all the orders supported by the tty_ I/O module, as well as those described below. All orders are supported when the I/O switch is open, except for event_info, which is supported when the I/O switch is attached.

event_info
: returns the name of the event channel over which wakeups are sent when input or status is received from the communications channel. The info_ptr must point to an aligned fixed binary (71) number, in which the value of the event channel is returned. This order should be used if the -async control argument appears in the attach description (see "Attach Description" above).

general_poll
: causes a general poll operation to be initiated at the 3270 controller. Once the I/O switch is open, either a general_poll order or a poll order (see below) must be issued before any input can be received; however, the general_poll order does not have to be repeated, as polling is automatically resumed when appropriate by the I/O module. The info_ptr is not used.

stop_general_poll
: causes automatic general polling to stop; polling is not resumed until a general_poll order is issued. The info_ptr is not used.

poll
: causes a specific poll operation to be performed on a single device connected to the controller. The info_ptr must point to a fixed binary number containing the identification number of the device to be polled. To ensure that the device is polled as soon as possible, this order usually should be preceded by a stop_general_poll order.

read
causes input or status information from a single device to be returned, if any is available. If no status or input is available for any device on the communications channel, then the process blocks if the -async control argument was not specified in the attach description; if it was specified, a status code of error_table_$request_pending is returned.

The info_ptr must point to a user-supplied structure of the following form:

```
dcl 1 read_ctl aligned,
      2 version fixed bin,
      2 areap ptr,
      2 read_infop ptr,
      2 max_len fixed bin,
      2 max_fields fixed bin;
```

where:

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

areap (Output)
is a pointer to an area in which the read_info structure (see below) is allocated.

read_infop (Output)
is a pointer to the read_info structure (see below).

max_len (Output)
is the largest number of characters that can be returned in a single data field (see below).

max_fields (Output)
is the largest number of data fields (see below) that can be returned in the read_info structure.

A read_info structure is allocated by the I/O module at the address specified by read_ctl.read_infop. This structure must be freed by the calling program. The read_info structure has the following form:

```
dcl 1 read_info aligned based (read_ctl.read_infop),
      2 version fixed bin,
      2 next_read_infop ptr,
      2 controller fixed bin,
      2 device fixed bin,
      2 reason,
        3 key fixed bin,
        3 sub_key fixed bin,
        3 code fixed bin(35),
      2 status,
        3 bits bit(12) unal,
        3 fill bit(24) unal,
      2 cursor_position fixed bin,
      2 max_fields fixed bin,
      2 max_len fixed bin,
      2 mod_fields fixed bin,
      2 data (read_ctl.max_fields refer (read_info.max_fields)),
        3 field_position fixed bin,
        3 contents char (read_ctl.max_len
                  refer (read_info.max_len)) var;
```

where:

version
: is the version number of this structure. The structure described here is version 1.

next_read_infop
: is a pointer to the next read_info structure used by the I/O module. (The calling program should not attempt to make use of this item.)

controller
: is the identification number of the 3270 controller from which the data or status has been received.

device
: is the identification number of the particular device (attached to the specified controller) that produced the data or status information.

reason
: describes the event that caused the structure to be filled in.

key
: identifies the nature of the event, which is either an error or status condition or an action on the part of the 3270 operator. It may have any of the following values:

  1 -- an error was detected at the device. A status code describing the error is returned in reason.code (see "code" below).

  2 -- the device reported status. The particular status is described by status.bits (see "status" below).

  3 -- the operator pressed the ENTER key.

  4 -- the operator pressed one of the program function (PF) keys. The particular key is identified by reason.sub_key (see "sub_key" below).

  5 -- the operator pressed one of the program attention (PA) keys. The particular key is identified by reason.sub_key (see "sub_key" below).

  6 -- the operator pressed the CLEAR key.

  7 -- the operator inserted a card in the identification card reader.

  8 -- the operator used the selector pen on an "attention" field.

  9 -- the operator pressed the TEST REQUEST key.

sub_key
: is the number of the PF or PA key pressed if reason.key is 4 or 5, respectively.

code
: is a status code describing an error at the device if reason.key is 1.

status
: contains the device status if reason.key is 2.

cursor_position
: is the current position of the cursor on the display screen.

max_fields
: is the number of elements in the data array (below).

max_len
: is the length of the longest contents string (below).

mod_fields
: is the number of elements in the data array (below) that are actually filled in in this instance of the structure.

data
: describes the data fields containing the input. No data fields are provided if reason.key is 1, 2, 5, or 6.

field_position
: is the starting buffer address of the data field.

contents
: is the contents of the data field. It is always a null string if reason.key is 8.

write
: causes commands and data to be sent to the 3270. The info_ptr must point to a user-supplied structure of the following form:

```
dcl 1 write_info aligned,
      2 version fixed bin,
      2 controller fixed bin,
      2 device fixed bin,
      2 from_device fixed bin,
      2 command fixed bin,
      2 write_ctl_char,
        3 bits unal,
          4 print_format bit(2) unal,
          4 start_printer bit(1) unal,
          4 sound_alarm bit(1) unal,
          4 keyboard_restore bit(1) unal,
          4 reset_mdt bit(1) unal,
        3 copy_bits bit(2) unal,
        3 pad bit(28) unal,
      2 max_fields fixed bin,
      2 max_len fixed bin,
      2 mod_fields fixed bin,
      2 data (max_write_fields
                     refer (write_info.max_fields)),
        3 orders unal,
          4 set_buffer_addr bit(1),
          4 start_field bit(1),
          4 insert_cursor bit(1),
          4 program_tab bit(1),
          4 repeat_to_addr bit(1),
          4 erase_to_addr bit(1),
        3 attributes unal,
          4 protected bit(1),
          4 numeric bit(1),
          4 display_form bit(2),
            4 reserved bit(1),
            4 mdt bit(1),
          3 pad1 bit(12) unal,
          3 field_position fixed bin,
          3 contents char (max_write_len
                        refer (write_info.max_len)) var;
```

where:

version
    is the version number of the structure. It must be 1.

controller
    is the identification number of the 3270 controller to which the data is to be sent.

device
    is the identification number of the device on that controller to which the data is to be sent.

from_device
    is the identification number of the device to be used as the "from" device for a copy command.

command
    is the command to be sent to the device. It may have any of the following values:

    1 -- write

    2 -- erase/write

    3 -- copy

    4 -- erase all unprotected

    5 -- read modified

    6 -- read buffer

write_ctl_char
    contains the low-order 6 bits of the write control character (WCC) to be inserted in the data stream. If command (above) is 3 (copy), this field contains the low-order 6 bits of the copy control character (CCC), except that the keyboard_restore and reset_mdt bits are replaced by the copy_bits (below).

copy_bits
    contains the two low-order bits of the copy control character if command (above) is 3 (copy). These are the bits that specify what type of data is to be copied.

max_fields
    is the number of elements in the data array (below).

max_len
    is the maximum length of any contents string (below).

mod_fields
    is the number of elements of the data array actually filled in in this instance of the structure.

data
: describes the individual data fields to be sent to the device.

orders
: identify orders to be inserted in the output stream.

set_buffer_addr
: indicates a set buffer address (SBA) order. The field_position (below) contains the buffer address to be set.

start_field
: indicates a start field (SF) order. The attribute character for the field is derived from attributes (below). If an SBA order is also indicated, the field starting address is contained in field_position (below); otherwise, the current device buffer address is used. The contents string, if nonnull, is written starting after the attribute character.

insert_cursor
: indicates an insert cursor (IC) order. If an SBA order is also indicated, the cursor is positioned to the address specified in field_position (below); otherwise it is set to the current device buffer address. If contents is nonnull, the data is written starting at the new cursor position.

program_tab
: indicates a program tab (PT) order. If an SBA order is also indicated, the tab is inserted at the address specified in field_position (below); otherwise it is inserted at the current device buffer address. If contents is nonnull, the data is written at the start of the field following the tab.

repeat_to_addr
: indicates a repeat to address (RA) order. The starting address is the current device buffer address; the ending address is specified in field_position (below). Neither an SBA order nor an EUA order can be indicated in the same field. The contents string must consist of a single character, which is to be repeated up to the address immediately preceding field_position.

erase_to_addr
: indicates an erase unprotected to address (EUA) order. The starting address is the current device buffer address; the ending address is specified in field_position (below). Neither an SBA order nor an RA order can be indicated in the same field. If contents is nonnull, the data is written starting at the address specified in field_position.

attributes
: contains the low-order 6 bits of the attribute character to be assigned to a field if start_field (above) is "1"b.

field_position
: is the device buffer address to be set if set_buffer_addr (above) is "1"b, or the ending address if repeat_to_addr or erase_to_addr (above) is "1"b.

contents
: is the data to be written. It may be a null string.

set_input_message_size
: specifies the length, in characters, of the largest input message that is expected. The info_ptr must point to a fixed binary number containing the message size. A size of 0 indicates that there is no maximum message size. Use of this order when a maximum message size is defined greatly increases the efficiency of the channel.

get_input_message_size
: is used to obtain the maximum input message size. The info_ptr must point to a fixed binary variable in which the maximum message size is returned as a result of the call. This size is the one most recently specified by a set_input_message_size order. If no set_input_message_size order has been done since the switch was attached, a size of 0 is returned.

Modes Operation

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

Name: ibm3780_

The ibm3780_ I/O module performs stream I/O to a remote I/O terminal that has the characteristics of an IBM 3780 data transmission terminal. The hardware options currently supported are defined by the control arguments described below.

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

This module in turn constructs an attach description for the module specified in the -comm control argument, passing the attach information for ascii or ebcdic, tty, transparent or nontransparent, and all other attach information specified by the caller.

Attach Description

```
ibm3780_ -control_args
```

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

-ascii
: transmits control information and data in ASCII.

-ebcdic
: converts control information and data to its EBCDIC representation before transmission. This is the default.

-multi_record
: transmits multiple records, up to 6, as a block, rather than separately. The default is single record transmission.

-physical_line_length N, -pll N
: sets the maximum character width of the remote I/O terminal printer to N characters. The default is 80 characters (120 if -device specifies printer). This variable is used to set tabs and pad records if the transparent option is specified.

-horizontal_tab, -htab
: supports tab control on the remote I/O terminal printer. Tabs are set every 10 spaces. The default is no tab control.

-tty STR
: connects the remote I/O station to the communications channel named STR.

-comm STR
: uses the communications I/O module specified by STR.

-transparent
: uses a transparent communication protocol.

-nontransparent
: uses a nontransparent communication protocol. This is the default.

-device STR
: specifies that this attachment is associated with the device STR.

-carriage_ctl STR
: the eight-character string STR, taken two characters at a time, sets the four carriage control characters which specify the advance of 0, 1, 2, and 3 lines. The default set of characters is ESCM, ESC/, ESCS, and ESCT where the mnemonic ESC means the ASCII escape character.

-slew_ctl STR
: the six-character string STR, taken two characters at a time, sets the slew control characters which specify top of form, inside page, and outside page. The default set of characters is ESCA, ESCA, and ESCA.

-printer_select STR
: the one-character string STR sets the printer select. The default printer select string is DC1.

-punch_select STR
: the one-character string STR sets the punch select. The default punch select string is DC2.

-terminal_type STR, -ttp STR
: STR specifies the terminal type whose conversion, translation, and special tables defined in the user or system terminal type table (TTT) are used to convert and translate input and output to and from the device. If not specified, no conversion or translation is performed. For more information about the allowable conversion values see "Notes" below.

Open Operation

The ibm3780_ I/O module supports stream_input, stream_output, and stream_input_output opening modes.

Put Chars Operation

The put_chars entry splits the data to be written into blocks of 80 or 512 characters, depending on whether multirecord mode is enabled, and transmits the number of characters specified to the specified communication I/O module. The blocks are of fixed or variable length, depending on whether transparent mode is enabled or not, respectively.

Get Chars Operation

The get_chars entry reads characters up to 80 or 512 characters, depending on whether multirecord mode is enabled, and returns the number requested, up to the next record separator.

Control Operation

This I/O module supports all the control operations supported by the communications I/O module specified in the attach description. In addition, it supports the following:

set_bsc_modes
: sets the character mode, either ascii or ebcdic and transparency. The input structure is defined as follows:

```
dcl 1 set_bsc_modes aligned,
      2 char_mode bit(1) unaligned,
      2 transparent bit(1) unaligned;
```

where char_mode = "1"b if ebcdic and "0"b if ascii, and transparent = "1"b if transparency is enabled and "0"b if not.

select_device
: selects the subdevice (either printer, punch, or teleprinter) to which output is next directed. The input structure is of the form:

```
dcl device char(32) based;
```

set_multi_record_mode
: sets the number of records per block. The input structure is of the form:

```
dcl record_number fixed bin based;
```

Modes Operation

This module supports the nonedited and default modes, which set and reset the edited output conversion, if it has been enabled by the -terminal_type control argument.

Notes

The only allowable values in the output conversion table are 00 and any values greater than 16. All values defined in the description of the tty_ I/O module are allowed for input conversion. Input and output translation may be up to 256 characters in length.

Name: remote_input_

The remote_input_ I/O module performs record input from a terminal I/O module which is assumed to be connected to a remote I/O device, such as a Honeywell Level 6 remote batch facility (G115 type), an IBM 2780, or an IBM 3780. Except for hardware restrictions, this module performs some code conversion and control in such a way that remote and local card reading are the same.

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

This module in turn constructs an attach description for the module specified in the -terminal control argument, passing the other attach information specified by the caller.

Attach Description

```
remote_input_ -control_args
```

where control_args may be chosen from the following:

-terminal STR
: STR specifies the terminal I/O module to be attached by this device I/O module. (required)

-device STR
: STR defines the device type which this I/O module is attempting to simulate. The acceptable values for STR are reader, printer_in and punch_in. This control argument is optional. If not supplied, a device type of reader is assumed.

-runout_spacing N, -runsp N
: This control argument is accepted and ignored for compatibility with other device-level I/O modules. It is not passed on to the terminal I/O module.

-physical_line_length N, -pll N
: This control argument is accepted and ignored for compatibility with other device-level I/O modules. It is not passed on to the terminal I/O module.

-record_len N
: defines the maximum record length (buffer size) for data from the terminal I/O module in characters. The accepted ranges are 80 to 160 for the device type of reader, and 10 to 1024 otherwise. If this control argument is not given, the maximum for the device type is assumed.

All other attach control arguments are assumed to belong to the terminal I/O module. These are passed on as part of its attach description. The -device option passed on to the terminal I/O module will specify one of the following devices: reader, printer, or punch. See the description of the terminal I/O module for a full definition of required and optional control arguments.

Open Operation

The remote input I/O module supports the stream_input opening mode. The terminal I/O module switch is in turn opened with the record_input mode.

Get Chars Operation

The get_chars entry reads one record from the terminal I/O module and returns up to the number of specified characters. If the number of characters in the record is greater than the requested number, error_table_$data_loss is returned along with the data.

Control Operation

The remote_input_ device I/O module supports the following control operations:

reset
: sets the current record count to 0 and passes the control operation on to the terminal I/O module.

get_count
: returns the current record count. This is the count of records read from the terminal I/O module since the last reset control operation. This operation is not passed on to the terminal I/O module.

  The info_pointer must point to the following structure. (This structure is taken from the counts structure in prt_order_info.incl.pl1 for compatibility with procedures which use several device I/O modules.)

```
dcl 1 counts aligned based,
    2 prt_data_pad (4) fixed bin,
    2 record_count fixed bin (35),
    2 prt_pad fixed bin;
```

  The variable record_count will contain the returned value. This corresponds with the variable line_count from the other structure.

All other control operations are passed on to the terminal I/O module.

Modes Operation

This I/O module supports the modes defined by the terminal I/O module specified in the attach description.

Name: remote_printer_

The remote_printer_ I/O module presents a stream I/O interface to the caller and performs record output to a printer, which is assumed to be part of a remote I/O device, such as a Honeywell Level 6 remote batch facility (G115 type), an IBM 2780, or an IBM 3780. Except for hardware restrictions, this module performs all the necessary code conversion and control in such a way that remote and local printing are the same.

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

This module in turn constructs an attach description for the module specified in the -terminal control argument, passing the attach information for horizontal tabbing, physical line length, and all other attach information specified by the caller.

Attach Description

remote_printer_ -control_args

where control_args are optional, with the exception of -terminal, and may be chosen from the following:

-physical_line_length N, -pll N
: printer has a maximum line width of N characters. The default is 132 characters.

-physical_page_length N, -ppl N
: printer has a maximum line count per page of N. The default is 66 lines.

-horizontal_tab, -htab
: printer has a horizontal tab feature. The default is no tab control.

-terminal STR
: uses the terminal I/O module specified by STR. This control argument is required.

Open Operation

The remote printer I/O module supports the stream_output opening mode.

Put Chars Operation

The put_chars entry converts a character string delimited by a newline character to an image suitable for printing and transmits this image to the terminal I/O module. This operation is repeated until all the characters specified by the caller have been transmitted.

Control Operation

This I/O module supports all the control operations supported by the terminal I/O module specified in the attach description. In addition, it supports all the printer control operations supported by the printer I/O module prtdim_ (see Appendix B).

Modes Operation

This I/O module supports all the modes supported by the terminal I/O module specified in the attach description. In addition, it supports all the modes supported by the printer I/O module prtdim_ (see Appendix B). It supports the two modes non_edited and default, which enable and disable edited output conversion, if output conversion has been enabled by the terminal I/O module.

Position Operation

This I/O module supports all the position operations supported by the terminal I/O module specified in the attach description.

Name: remote_punch_

The remote_punch_ I/O module presents a stream I/O interface to the caller and performs record output to a card punch, which is assumed to be part of a remote I/O device, such as a Honeywell Level 6 remote batch facility (G115 type), an IBM 2780, or an IBM 3780. Except for hardware restrictions, this module performs all the necessary code conversion and control in such a way that remote and local card punching are the same.

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

This module in turn constructs an attach description for the module specified in the -terminal control argument, passing the other attach information specified by the caller.

Attach Description

```
remote_punch_ -control_arg
```

where control_args may be chosen from the following:

-terminal STR
: STR specifies the terminal I/O module to be attached to this device I/O module. (Required)

-device STR
: defines the type of device to be simulated by this I/O module and may be either "punch" or "reader_simulator". This specification is passed to the terminal I/O module as "-device punch" or "-device reader", respectively. (Default is "punch".)

-card_ll N
: specifies the length of records (cards) supported by the terminal I/O module. (Default is 80.)

-non_edited
: specifies that non-printing characters may be passed directly to the terminal I/O module. (Default is that these characters are not passed.)

-horizontal_tab, -htab
: specifies that the device supports the horizontal tab character. (Default is the use of the appropriate number of spaces.)

-runout_spacing N, -rnsp N
-physical_page_length N, -ppl N
: are accepted and ignored for compatibility with other device I/O modules.

All other attach arguments are passed directly to the terminal I/O module.

Open Operation

The remote punch I/O module supports the stream_output opening mode.

Put Chars Operation

The put_chars entry splits the data to be written into records of the size given by -card_ll and transmits these records to the terminal I/O module. This operation is repeated until all the characters specified by the caller have been transmitted.

Control Operation

The remote_punch device I/O module supports the following control operations:

reset
: sets the current record count to zero, returns to punching in RMCC (remote Multics card code), and passes the order to the terminal I/O module.

get_count
: returns the current record count which is the number of records written to the terminal I/O module since the last reset control operation. This operation is not passed on to the terminal I/O module. The info_ptr must point to the following PL/1 structure. (This structure is taken from the counts structure in prt_order_info.incl.pl1 for compatibility with procedures which use several device I/O modules.)

```
dcl 1 counts aligned based,
      2 prt_data_pad (4) fixed bin,
      2 record_count fixed bin (35),
      2 prt_pad fixed bin;
```

The variable record_count will contain the returned value. This corresponds with the variable line_count from the other structure.

binary_punch
: requests that all subsequent data be punched in binary (rather than RMCC) if supported by the terminal I/O module. This control order is then passed on to the terminal I/O module.

All other control operations are passed directly to the terminal I/O module for processing.

Modes Operation

This I/O module supports the RMCC output card mode defined in Section V of the MPM Reference Guide. It also supports the two modes non_edited and default, which enable and disable edited output conversion, if output conversion has been enabled by the terminal I/O module.

Position Operation

This I/O module supports all the position operations supported by the terminal I/O module specified in the attach description.

Name: remote_teleprinter_

The remote_teleprinter_ I/O module presents a stream I/O interface to the caller and performs record I/O to a terminal or printer, which is assumed to be part of a remote I/O device, such as a Honeywell Level 6 remote batch facility (G115 type), an IBM 2780, or an IBM 3780.

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

This module in turn constructs an attach description for the module specified in the -terminal control argument, passing the attach information for ASCII or EBCDIC, horizontal tabbing, physical line length, and all other attach information specified by the caller.

Attach Description

```
remote_teleprinter_ -control_args
```

where control_args are optional, with the exception of -terminal, and may be chosen from the following:

-physical_line_length N, -pll N
: output device has a maximum line width of N characters. The default is 80 characters.

-physical_page_length N, -ppl N
: output device has a maximum line count per page of N. The default is 66 lines.

-horizontal_tab, -htab
: output device has a horizontal tab feature. The default is no tab control.

-terminal STR
: uses the terminal I/O module specified by STR. This control_arg is required.

-runout_spacing N, -runsp N
: outputs N newline characters with each runout operation. This allows the operator to see messages still under the printer mechanism for terminals which have only a printer as an output device. The default is 0.

Open Operation

The remote_teleprinter_ I/O module supports the stream_input_output opening mode.

Put Chars Operation

The put_chars entry converts a character string ending in a newline character to an image suitable for printing and transmits this image to the terminal I/O module.

Get Chars Operation

The get_chars entry reads the number of specified characters from the terminal I/O module.

Get Line Operation

The get_line entry reads one record from the terminal I/O module, appends a new line, and returns as many characters as requested by the caller, or the whole record if it is shorter. If the record is longer than requested, error_table_$data_loss is returned.

Control Operation

This I/O module supports all the control operations supported by the terminal I/O module specified in the attach description. In addition, it supports all the printer control operations supported by the printer I/O module prtdim_ (see Appendix B).

Modes Operation

This I/O module supports all the modes supported by the terminal I/O module specified in the attach description. In addition, it supports all the modes supported by the printer I/O module prtdim_ (see Appendix B). It also supports the two modes non_edited and default, which enable and disable edited output conversion if output conversion has been enabled by the terminal I/O module.

Position Operation

This I/O module supports all the position operations supported by the terminal I/O module specified in the attach description.

Name: tty_

The tty_ I/O module supports I/O from/to devices that can be operated in a typewriter-like manner, e.g., the user's terminal.

Entry points in the module are not called directly by users; rather the module is accessed through the I/O system. See "Multics Input/Output System" in Section 5 of the MPM Reference Guide for a general description of the I/O system.

Attach Description

tty_ {device} {-control_args}

where device is the channel name of the device to be attached (channel names are described in Appendix A). If a device is not given, the -login_channel control argument must be given. The star convention is allowed.

Control arguments may be chosen from the following:

-login_channel
: specifies attachment to the user's primary login channel. If a device is not specified then the user's login channel is used. This control argument flags this switch for reconnection by the process disconnection facility. If the user's login device should hang up, this switch will be automatically closed, detached, attached and opened on the user's new login channel when the user reconnects, if permission to use this facility is specified in the SAT and PDT for the user.

-destination DESTINATION
: this control argument specifies that the attached device is to be called using the address DESTINATION. In the case of telephone auto_call lines, DESTINATION is the telephone number to be dialed. Use of this control argument requires the dialok attribute.

-dial_id STR
: specifies that dial connections are to be accepted on the dial_id STR. Use of this control argument requires the dialok attribute. The dial command is then used to connect a terminal on the dial_id STR. If STR is not a registered dial_id, then the Person_id.Project_id of the process being connected to must be supplied to the dial command. For example:

```
dial STR Person.Project
```

To become a registered server, the process must have rw access to >scq>rcp>dial.STR.acs.

-no_block
: specifies that the device is to be managed asynchronously. tty_ will not block to wait for input to be available or output space to be available. (See "Buffering" below for more details.) This control argument should not be used on the login channel, because it will cause the command listener to loop calling get_line.

-no_hangup_on_detach
: prevents the detach entrypoint from hanging up the device. This is not meaningful for the login channel.

-hangup_on_detach
: causes the detach entrypoint to hang up the device automatically. This is not meaningful for the login channel.

-resource STR
: specifies the desired characteristics of a channel. STR (which can be null) consists of reservation attributes separated by commas. The channel used by a dial-out operation must have the characteristics specified in the reservation string. Reservation attributes consist of a keyword and optional argument. Attributes allowed are:

```
baud_rate=BAUD_RATE
line_type=LINE_TYPE
```

where BAUD_RATE is a decimal representation of the desired channel line speed and LINE_TYPE is a valid line type, chosen from line_types.incl.pl1 (see set_line_type, below).

This page intentionally left blank.

Notes

The device specified must be available to the attaching process. The user's login device is always available. Any devices acquired with the dial_manager_ subroutine are also available. If the device is in slave service, and the user has appropriate access to its access control segment (rw to >sc1>rcp>NAME.acs), tty_ will attempt to make it available using the privileged_attach mechanism of dial_manager_. If the -destination control argument is specified, the dial_out mechanism is used (the user must have rw access to >sc1>rcp>NAME.acs). If the -dial_id control argument is specified, the allow_dials or registered server mechanism is used. See the documentation of dial_manager_ in the MPM Subsystem Writer's Guide for more details.

Opening

The opening modes supported are stream_input, stream_output, and stream_input_output.

Editing

On both input and output, data is automatically edited as described in "Typing Conventions" in Section 2. To control the editing, use the modes operation. Details on the various modes are given below.

Buffering

This I/O module will block to await either the availability of input characters or the availability of output buffer space, unless the -no_block control argument is specified in the attach description. If the -no_block attach description control argument is specified, the behavior of the iox_$put_chars, iox_$get_chars and iox_$get_line calls changes. If the put_chars entrypoint cannot write all the characters supplied, it will return a nonstandard status code consisting of the negative of the number of characters actually written plus one (-(n_chars_written +1)). Any positive status code should be interpreted as a standard system status code. The get_chars and get_line entrypoints will return zero status codes and zero characters read if there is no input available.

Interrupted Operations

When an I/O operation (except detach) being performed on a switch attached by this I/O module is interrupted by a signal, other operations may be performed on the switch during the interruption. If the interrupted operation is get_line, get_chars or put_chars, and another get_line, get_chars or put_chars operation is performed during the interruption, the "start" control operation should be issued before the interrupted operation is resumed.

Get Chars Operation

The get_chars operation reads as many characters as are available, up to, but not exceeding, the number requested by the caller. No error code is returned if the number of characters read is less than the number requested. At least one character is always returned (unless the number requested is zero). The characters read may comprise only a partial input line, or may comprise several input lines; no assumptions can be made in this regard.

Get Line Operation

The get_line operation is supported. No error code is returned if the read operation occurs with the input buffer length at zero. For further explanation, see the iox_$get_line entry in MPM Subroutines.

Put Chars Operation

The put_chars operation is supported. For further explanation, see the iox_$put_chars entry in MPM Subroutines.

Control Operation

The following orders are supported when the I/O switch is open. Except as noted, the info_ptr should be null. The orders are divided into categories. Local orders perform a specific function one time only; global orders change the way the system interfaces with the terminal; and other orders fit in neither category. Control orders are performed through the iox_$control entry, as described in MPM Subroutines.

LOCAL

abort
    flushes the input and output buffers.

interrupt
    sends an out-of-band interrupt signal (quit signal) to the terminal.

resetread
    flushes the input buffer.

resetwrite
    flushes the output buffer.

hangup
    disconnects the telephone line connection of the terminal, if possible. This makes the terminal unavailable for further use.

listen
: sends a wakeup to the process once the line associated with this device identifier is dialed up.

printer_off
: causes the printer mechanism of the terminal to be temporarily disabled if it is physically possible for the terminal to do so; if it is not, the status code error_table_$action_not_performed is returned (see "Notes" below).

printer_on
: causes the printer mechanism of the terminal to be reenabled (see "Notes" below).

wru
: initiates the transmission of the answerback of the device, if it is so equipped. This operation is allowed only for the process that originally attached the device (generally the initializer process). The answerback may subsequently be read by means of the get_chars input/output operation.

start_xmit_hd
: causes the channel to remain in a transmitting state at the completion of the next block of output, rather than starting to accept input. The line will then remain in a transmitting state until the stop_xmit_hd control operation is issued. This operation is valid only for terminals with line type LINE_ARDS.

stop_xmit_hd
: causes the channel to resume accepting input from the terminal (after the completion of current output, if any). This operation is only valid for ARDS-like terminals and is used only to counteract a preceding start_xmit_hd operation.

GLOBAL

set_line_type
: sets the line type associated with the terminal to the value supplied. The info_ptr should point to a fixed binary variable containing the new line_type. Line types can be any of the following named constants defined in the include file line_types.incl.pl1:

  LINE_ASCII
  : device similar to 7-bit ASCII using Bell 103-type modem protocol

  LINE_1050
  : device similar to IBM Model 1050

LINE_2741
device similar to IBM Model 2741, with or without auto EOT inhibit

LINE_ARDS
device similar to Adage, Inc. Advanced Remote Display Station (ARDS) protocol using Bell 202C6-type modem

LINE_SYNC
synchronous connections, no protocol

LINE_G115
ASCII synchronous connection, Model G-115 remote computer protocol

LINE_BSC
binary synchronous protocol

LINE_ETX
device similar to TermiNet 1200 protocol using Bell 202C5-type modem

LINE_VIP
device similar to Honeywell Model 7700 Visual Information Projection (VIP) standalone terminal

LINE_ASYNC1
LINE_ASYNC2
LINE_ASYNC3
site-supplied asynchronous protocols

LINE_SYNC1
LINE_SYNC2
LINE_SYNC3
site-supplied synchronous protocols

LINE_POLLED_VIP
device similar to Honeywell Model 7700 Visual Projection System (VIP) polled terminal concentrator subsystem.

LINE_X25LAP
X.25 network connection using the link access protocol (LAP)

LINE_COLTS
special software channel used for Communications Online Test and Diagnostics System

This operation is not permitted while the terminal is in use.

refuse_printer_off
causes subsequent printer_off and printer_on orders to be rejected except when in echoplex mode.

accept_printer_off
causes subsequent printer_off and printer_on orders to be accepted if possible.

set_delay
sets the number of delay characters associated with the output of carriage-motion characters. The info_ptr points to the following structure: (defined in tty_convert.incl.pl1)

```
dcl 1 delay_struc     based aligned,
      2 version       fixed bin,
      2 default       fixed bin,
      2 delay,
        3 vert_nl     fixed bin,
        3 horz_nl     float bin,
        3 const_tab   fixed bin,
        3 var_tab     float bin,
        3 backspace   fixed bin,
        3 vt_ff       fixed bin;
```

where:

version
is the version number of the structure. It must be 1.

default
indicates, if nonzero, that the default values for the current terminal type and baud rate are to be used and that the remainder of the structure is to be ignored.

vert_nl
is the number of delay characters to be output for all newlines to allow for the linefeed ($-127 \leq$ vert_nl ≤ 127). If it is negative, its absolute value is the minimum number of characters that must be transmitted between two linefeeds (for a device such as a TermiNet 1200).

horz_nl
is a number to be multiplied by the column position to obtain the number of delays to be added for the carriage return portion of a newline ($0 \leq$ horz_nl ≤ 1). The formula for calculating the number of delay characters to be output following a newline is:

```
ndelays=vert_nl+fixed(horz_nl*column)
```

const_tab
is the constant portion of the number of delays associated with any horizontal tab character ($0 \leq$ const_tab ≤ 127).

var_tab
is the number of additional delays associated with a horizontal tab for each column traversed ($0 \leq$ var_tab ≤ 1). The formula for calculating the number of delays to be output following a horizontal tab is:

```
ndelays = const_tab + fixed (var_tab*n_columns)
```

backspace
is the number of delays to be output following a backspace character ($-127 \leq$ backspace ≤ 127). If it is negative, its absolute value is the number of delays to be output with the first backspace of a series only (or a single backspace). This is for terminals such as the TermiNet 300 which need delays to allow for hammer recovery in case of overstrikes, but do not require delays for the carriage motion associated with the backspace itself.

vt_ff
    is the number of delays to be output following a vertical tab or formfeed ($0 \leq$ vt_ff ≤ 511).

get_delay
    is used to find out what delay values are currently in effect. The info_ptr points to the structure described for set_delay (above), which is filled in as a result of the call (except for the version number, which must be supplied by the caller).

set_editing_chars
    changes the characters used for editing input. The info_ptr points to the following structure:

```
dcl 1 editing_chars  aligned,
      2 version      fixed bin,
      2 erase        char(1) unaligned,
      2 kill         char(1) unaligned;
```

where:

version
    is the version number of this structure. It must be 2.

erase
    is the erase character.

kill
    is the kill character.

The following rules apply to editing characters:

1. The two editing characters may not be the same.

2. No carriage-movement character (carriage return, newline, horizontal tab, backspace, vertical tab, or formfeed) may be used for either of the editing functions.

3. NUL and space may not be used for either editing function.

4. If either of the editing characters is an ASCII control character, it will not have the desired effect unless ctl_char mode is on (see "Modes Operation" below).

get_editing_chars
    is used to find out what input editing characters are in effect. The info_ptr points to the structure described above for set_editing_chars, which is filled in as a result of the call (except for the version number, which must be supplied by the caller).

set_input_translation
provides a table to be used for translation of terminal input to ASCII. The info_ptr points to a structure of the following form: (defined in tty_convert.incl.pl1)

```
dcl 1 cv_trans_struc   aligned,
      2 version        fixed bin,
      2 default        fixed bin,
      2 cv_trans       aligned,
        3 value        (0 : 255) char(1) unaligned;
```

where:

version
is the version number of the structure. It must be 1.

default
indicates, if nonzero, that the default table for the current terminal type is to be used, and the remainder of the structure is ignored.

values
are the elements of the table. This table is indexed by the value of a typed input character, and the corresponding entry contains the ASCII character resulting from the translation. If the info_ptr is null, no translation is to be done.

NOTE: In the case of a terminal that inputs 6-bit characters and case-shift characters, the first 64 characters of the table correspond to characters in lower shift, and the next 64 correspond to characters in upper shift.

set_output_translation
provides a table to be used for translating ASCII characters to the code to be sent to the terminal. The info_ptr points to a structure like that described for set_input_translation (above). The table is indexed by the value of each ASCII character, and the corresponding entry contains the character to be output. If the info_ptr is null, no translation is to be done.

NOTE: For a terminal that expects 6-bit characters and case-shift characters, the 400(8) bit must be turned on in each entry in the table for a character that requires upper shift and the 200(8) bit must be on in each entry for a character that requires lower shift.

set_input_conversion
provides a table to be used in converting input to identify escape sequences and certain special characters. The info_ptr points to a structure of the following form: (defined in tty_convert.incl.pl1)

```
dcl 1 cv_trans_struc   aligned,
      2 version        fixed bin,
      2 default        fixed bin,
      2 cv_trans       aligned,
        3 value        (0 : 255) fixed bin(8) unaligned;
```

where version, default, and value are as described in the cv_trans_struc structure used with the set_input_translation order above. The table is indexed by the ASCII value of each input character (after translation, if any), and the corresponding entry contains one of the following values: (Mnemonic names for these values are defined in tty_convert.incl.pl1)

0 -- ordinary character
1 -- break character
2 -- escape character
3 -- character to be thrown away
4 -- formfeed character (to be thrown away if page length is nonzero)
5 -- this character and immediately following character to be thrown away

set_output_conversion
provides a table to be used in formatting output to identify certain kinds of special characters. The info_ptr points to a structure like that described for set_input_conversion (above). The table is indexed by each ASCII output character (before translation, if any), and the corresponding entry contains one of the following values: (Mnemonic names for these values are defined in tty_convert.incl.pl1)

0 -- ordinary character
1 -- newline
2 -- carriage return
3 -- horizontal tab
4 -- backspace
5 -- vertical tab
6 -- formfeed
7 -- character requiring octal escape
8 -- red ribbon shift
9 -- black ribbon shift
10 -- character does not change the column position
11 -- this character together with the following one do not change the column position (used for hardware escape sequences)
12 -- character is not to be sent to the terminal
17 or greater -- a character requiring a special escape sequence. The indicator value is the index into the escape table of the sequence to be used, plus 16. The escape table is part of the special characters table; see the set_special order below.

get_input_translation
get_output_translation
get_input_conversion
get_output_conversion
These orders are used to obtain the current contents of the specified table. The info_ptr points to a structure like the one described for the corresponding "set" order above, which is filled in as a result of the call (except for the version number, which must be supplied by the caller). If the specified table does not exist (no translation or conversion is required), the status code error_table_$no_table is returned.

set_special
provides a table that specifies sequences to be substituted for certain output characters, and characters that are to be interpreted as parts of escape sequences on input. Output sequences are of the following form: (defined in tty_convert_incl.pl1)

```
dcl 1 c_chars     based aligned,
      2 count     fixed bin(8) unaligned,
      2 chars(3)  char(1) unaligned;
```

where:

count
is the actual length of the sequence in characters ($0 \le count \le 3$). If count is zero, there is no sequence.

chars
are the characters that make up the sequence.

The info_ptr points to a structure of the following form: (defined in tty_convert_incl.pl1)

```
dcl 1 special_chars_struc        aligned based,
    2 version                    fixed bin,
    2 default                    fixed bin,
    2 special_chars
      3 nl_seq                   aligned like c_chars,
      3 cr_seq                   aligned like c_chars,
      3 bs_seq                   aligned like c_chars,
      3 tab_seq                  aligned like c_chars,
      3 vt_seq                   aligned like c_chars,
      3 ff_seq                   aligned like c_chars,
      3 printer_on               aligned like c_chars,
      3 printer_off              aligned like c_chars,
      3 red_ribbon_shift         aligned like c_chars,
      3 black_ribbon_shift       aligned like c_chars,
      3 end_of_page              aligned like c_chars,
      3 escape_length            fixed bin,
      3 not_edited_escapes       (sc_escape_len refer
                                   (special_chars.escape_length))
                                   like c_chars,
      3 edited_escapes           (sc_escape_len refer
                                   (special_chars.escape_length))
                                   like c_chars,
      3 input_escapes            aligned,
        4 len                    fixed bin(8) unaligned,
        4 str                    char (sc_input_escape_len refer
                                   (special_chars.input_escapes.len))
                                   unaligned,
      3 input_results            aligned,
        4 pad                    bit(9) unaligned,
        4 str                    char (sc_input_escape_len refer
                                   (special_chars.input_escapes.len))
                                   unaligned;
```

where:

version
    is the version number of this structure. It must be 1.

default
    is as above in set_input_translation.

nl_seq
    is the output character sequence to be substituted for a newline character. The nl_seq.count generally should be nonzero.

cr_seq
    is the output character sequence to be substituted for a carriage-return character. If count is zero, the appropriate number of backspaces is substituted. Either cr_seq.count or bs_seq.count (below), however, should be nonzero (i.e., both should not be zero).

bs_seq
    is the output character sequence to be substituted for a backspace character. If count is zero, a carriage return and the appropriate number of spaces are substituted. Either bs_seq.count or cr_seq.count (above), however, should be nonzero (i.e., both should not be zero).

tab_seq
    is the output character sequence to be substituted for a horizontal tab. If count is zero, the appropriate number of spaces is substituted.

vt_seq
    is the output character sequence to be substituted for a vertical tab. If count is zero, no characters are substituted.

ff_seq
    is the output character sequence to be substituted for a formfeed. If count is zero, no characters are substituted.

printer_on
    is the character sequence to be used to implement the printer_on control operation. If count is zero, the function is not performed.

printer_off
    is the character sequence to be used to implement the printer_off control operation. If count is zero, the function is not performed.

red_ribbon_shift
    is the character sequence to be substituted for a red ribbon-shift character. If count is zero, no characters are substituted.

black_ribbon_shift
is the character sequence to be substituted for a black ribbon-shift character. If count is zero, no characters are substituted.

end_of_page
is the character sequence to be printed to indicate that a page of output is full. If count is zero, no additional characters are printed and the cursor is left at the end of the last line.

escape_length
is the number of output escape sequences in each of the two escape arrays.

not_edited_escapes
is an array of escape sequences to be substituted for particular characters if the terminal is in "^edited" mode. This array is indexed according to the indicator found in the corresponding output conversion table (see the description of the set_output_conversion order above).

edited_escapes
is an array of escape sequences to be used in edited mode. It is indexed in the same fashion as not_edited_escapes.

input_escapes
is a string of characters each of which forms an escape sequence when preceded by an escape character (see the discussion of escape sequences in Section 2 for more detailed information).

input_results
is a string of characters each of which is to replace the escape sequence consisting of an escape character and the character occupying the corresponding position in input_escapes (above).

get_special
is used to obtain the contents of the special_chars table currently in use. The info_ptr points to the following structure (defined in tty_convert.incl.pl1):

```
dcl 1 get_special_info_struc  aligned,
      2 area_ptr              ptr,
      2 table_ptr             ptr;
```

where:

area_ptr (Input)
points to an area in which a copy of the current special_chars table is returned.

table_ptr (Output)
is set to the address of the returned copy of the table.

set_term_type
: sets the terminal type associated with the channel to one of the types defined in the terminal type table. The info_ptr should point to the following structure:

```
dcl 1 set_term_type_info      aligned,
      2 version               fixed bin,
      2 name                  char(32) unaligned,
      2 flags,
        3 initial_string      bit(1) unaligned,
        3 modes               bit(1) unaligned,
        3 ignore_line_type    bit(1) unaligned,
        3 mbz                 bit(33);
```

where:

version
: is the version number of the above structure. It must be 1.

name
: is the name of the terminal type to be set.

initial_string
: is "1"b if the initial string for the terminal type is to be transmitted to the terminal; otherwise, it is "0"b.

modes
: is "1"b if the default modes for the terminal type are to be set; otherwise it is "0"b.

ignore_line_type
: is "1"b if the terminal type to be set need not be compatible with the line type; otherwise it is "0"b.

mbz
: must be "0"b.

set_framing_chars
: specifies the pair of characters that the terminal generates surrounding input transmitted as a block or "frame". These characters must be specified in the character code used by the terminal. This order must be used for blk_xfer mode (see below) to be effective. The info_ptr must point to a structure with the following format:

```
dcl 1 framing_chars aligned,
    2 frame_begin char(1) unaligned,
    2 frame_end char(1) unaligned;
```

get_framing_chars
: causes the framing characters currently in use to be returned (see the set_framing_chars order, above). If no framing characters have been supplied, NUL characters are returned. The info_ptr must point to a structure like the one described for the set_framing_chars order; this structure is filled in as a result of the call.

set_wakeup_table
specifies a wakeup table, i.e. a set of wakeup characters, that controls the dispatching of input wakeups. The wakeup table operates in conjunction with wake_tbl mode. The wakeup table has no effect until wake_tbl mode is enabled. Once enabled, the standard method of generating input wakeups (normally one wakeup for each line) is suspended. Thereafter, wakeups are only generated when wakeup characters are received or when the buffer gets too full. The wakeup table cannot be changed while wake_tbl mode is enabled. The info_ptr should point to the following structure:

```
dcl 1 set_wakeup_table_info     aligned,
      2 version                 fixed bin,
      2 new_table,
        3 new_wake_map          (0:127) bit(1) unal,
        3 mbz1                  bit(16) unal,
      2 old_table,
        3 old_wake_map          (0:127) bit(1) unal,
        3 mbz2                  bit(16) unal;
```

where:

version (Input)
is the version number of this structure. It must be 1.

new_wake_map (Input)
is an array having one entry for each character in the ASCII character set. A value of "1"b defines a wakeup character. All other entries must be "0"b. If all entries are "0"b, the current wakeup table, if any, is deleted.

mbz1 (Input)
must be "0"b.

old_wake_map (Output)
is set to the value of the current wakeup table that is being replaced. If no current wakeup table exists, all entries are set to "0"b.

mbz2 (Output)
is set to "0"b.

The primary application for the wakeup table mechanism will be to reduce overhead incurred by text editors, such as qedx, while in input mode. While in input mode, a user process must wake up for each line of input even though no processing is immediately required. In wake_tbl mode, a process will only be awoken when input mode is exited or a large amount of input has been accumulated. However, since wake_tbl mode will cause more input to be buffered in ring 0 than before, a quit signal is likely to discard more input than before. If a user does not wish to lose input, he simply should avoid quitting while in input mode.

If a user does quit out of input mode, he will not remain in wake_tbl mode (under normal circumstances). The default modes established by the standard quit handler include ^wake_tbl. A start command will restore wake_tbl mode.

input_flow_control_chars
specifies the character(s) to be used for input flow control for terminals with line speed input capability. The terminal must be in iflow mode for the feature to take effect. (See the discussion of flow control in Section 2.) The info_ptr must point to a structure with the following format:

```
dcl 1 input_flow_control_info   aligned,
      2 suspend_seq             unaligned,
        3 count                 fixed bin(9) unsigned,
        3 chars                 char(3),
      2 resume_seq              unaligned,
        3 count                 fixed bin(9) unsigned,
        3 chars                 char(3),
      2 timeout                 bit(1);
```

where:

suspend_seq
is the character sequence that the system sends to tell the terminal to stop sending input, or that the terminal sends to inform the host that it is suspending input. count is an integer from 0 to 3 that specifies the number of characters in the sequence. chars are the characters themselves. At present, only sequences of length 0 or 1 are supported.

resume_seq
is the character sequence to be sent by the system to the terminal to tell it to resume transmission of input. count and chars are as above.

timeout
is "1"b if the resume character is to be sent to the terminal after input has ceased for one second, whether or not a suspend character has been received.

output_flow_control_chars
enables either of two output flow control protocols and specifies the characters to be used for output flow control. The terminal must be in oflow mode for the feature to take effect. (See the discussion of flow control in Section 2.) The info_ptr must point to a structure with the following format:

```
dcl 1 output_flow_control_info  aligned,
      2 flags                   unaligned,
        3 suspend_resume        bit(1),
        3 block_acknowledge     bit(1),
        3 mbz                   bit(16),
      2 buffer_size             fixed bin(18) unsigned unaligned,
      2 suspend_or_etb_seq      unaligned,
        3 count                 fixed bin(9) unsigned,
        3 chars                 char(3),
      2 resume_or_ack_seq       unaligned,
        3 count                 fixed bin(9) unsigned,
        3 chars                 char(3);
```

where:

suspend_resume
is "1"b to specify a suspend/resume protocol.

block_acknowledge
is "1"b to specify a block acknowledgement protocol.

buffer_size
is the number of characters in the terminal's buffer if block_acknowledge is "1"b. Otherwise it is ignored.

suspend_or_etb_seq
is the character sequence sent by the terminal to tell the system to suspend output if suspend_resume is "1"b, or the end_of_block character sequence if block_acknowledge is "1"b. count and chars are as described for the input_flow_control_chars order above.

resume_or_ack_seq
is the character sequence sent by the terminal to indicate that output may be resumed if suspend_resume is "1"b, or the character sequence sent by the terminal to acknowledge completion of a block if block_acknowledge is "1"b. count and chars are as above.

get_ifc_info
causes the characters currently in use for input flow control to be returned (see the input_flow_control_chars order, above). The info_ptr must point to a structure like the one described for the input_flow_control_chars order, which will be filled in as a result of the call. If no characters are currently set, the count fields are set to 0.

get_ofc_info
causes the characters and protocol currently in use for output flow control to be returned (see the output_flow_control_chars order, above). The info_ptr must point to a structure like the one described for the output_flow_control_chars order, which will be filled in as a result of the call. If no output flow control protocol is currently in use, the count fields are set to 0 and both suspend_resume and block_acknowledge are set to "0"b.

get_channel_info
returns the name of the attached channel and its hardcore device index. The info_ptr must point to the following structure (defined in tty_get_channel_info.incl.pl1):

```
dcl 1 tty_get_channel_info          aligned based,
      2 version                     fixed bin,
      2 devx                        fixed bin,
      2 channel_name                char (32);
```

where:

1. version (Input)
   is the version of this structure. It must be set to tty_get_channel_info_version.

2. devx (Output)
   is the hardcore device index for the channel.

3. channel_name (Output)
   is the name of the channel.

OTHER

read_status
: tells whether or not there is any type-ahead input waiting for a process to read. The info_ptr should point to the following structure, (defined in tty_read_status_info.incl.pl1) which is filled in by the call:

```
          dcl 1 tty_read_status_info aligned based,
                2 event_channel fixed bin (71),
                2 input_pending bit (1);
```

where:

ev_chan
: is the event channel used to signal the arrival of input.

input_available
: indicates whether input is available.
"0"b no input
"1"b input

write_status
: tells whether or not there is any write-behind output that has not been sent to the terminal. The info_ptr should point to the following structure, which is filled in by the call:

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

where:

ev_chan
: is the event channel used to signal the completion of output.

output_pending
: indicates whether output is pending.
"0"b no output
"1"b output

quit_enable
: causes quit signal processing to be enabled for this device. (Quit signal processing is initially disabled.)

quit_disable
: causes quit signal processing to be disabled for this device.

start
: causes a wakeup to be signalled on the event channel associated with this device. This request is used to restart processing on a device whose wakeups may have been lost or discarded.

store_id
: stores the answerback identifier of the terminal for later use by the process. The info_ptr should point to a char(4) variable that contains the new identifier.

terminal_info
    returns information about the terminal. The info_ptr should point to the following structure:

```
          dcl 1 terminal_info            aligned,
                2 version                fixed bin,
                2 id                     char(4) unaligned,
                2 term_type              char(32) unaligned,
                2 line_type              fixed bin,
                2 baud_rate              fixed bin,
                2 reserved (4)           fixed bin;
```

where:

version (Input)
    is the version number of the above structure. It must be 1.

id (Output)
    is the terminal identifier derived from the answerback.

term_type (Output)
    is the terminal type name.

line_type (Output)
    is the line type number.

baud_rate (Output)
    is the baud rate at which the terminal is running.

reserved
    is reserved for future use.

send_initial_string *
    transmits an initialization string to the terminal in raw output (rawo) mode. Due to the use of raw output mode, the string must comprise character codes recognized by the terminal. If the info_ptr is null, the initial string defined for the terminal type is used. Otherwise, the info_ptr should point to the following structure:

```
          dcl 1 send_initial_string_info         aligned,
                2 version                        fixed bin,
                2 initial_string                 char(512) varying;
```

where:

version
    is the version number of the above structure. It must be 1.

initial_string
    is the initial string to be sent.

set_default_modes
    sets the modes to the default modes for the terminal type.

set_event_channel
    specifies the ipc_ event channel that will receive wakeups for this attachment. Wakeups are received for input available, output completed, and state changes such as hangups and quits. The channel may be event wait or event call. If it is event call, the -no_block control argument must be present in the attach description for correct operation.

The info_pointer should point to a fixed bin (71) aligned quantity containing a valid ipc_ channel identifier. No check for the validity of the channel is made. If the channel is invalid, incorrect operation will result.

If this control order is not given before the opening of the switch, tty_ will attempt to allocate a fast event channel. Fast event channels may not be converted to call channels and receive no associated message. If tty_ cannot allocate a fast channel, an ordinary event wait channel will be created and used. This control order is accepted while the switch is closed or open. If the switch is open, the new channel replaces the old one.

get_event_channel
: returns the identifier of the ipc_ event channel associated with the channel. The info_pointer should point to a fixed bin (71) aligned quantity into which the channel identifier will be stored. If the switch is not yet open and the set_event_channel order has not been given, the result will be zero.

This control order, which replaces the event_info control order, is accepted with the switch open or closed. For more information on event management, see the set_event_channel control order.

copy_meters
: causes the current cumulative meters associated with the channel to be copied to unwired storage, so that the statistics for the channel can be determined both for the life of the system and for the current dialup. This order can only be issued by the "owning" process (normally the initializer). The info_ptr should be null.

get_meters
: causes current values of meters associated with the channel to be returned. The info_ptr must point to a structure of the following form, defined in the include file get_comm_meters_info.incl.pl1:

```
dcl 1 get_comm_meters_info aligned based,
      2 version fixed bin,
      2 pad fixed bin,
      2 subchan_ptr pointer,
      2 logical_chan_ptr pointer,
      2 parent_ptr pointer,
      2 subchan_type fixed bin,
      2 parent_type fixed bin;
```

where:

version (Input)
: must be 1.

subchan_ptr (Input)
: is a pointer to a structure in which multiplexer-specific meters kept at the subchannel level are to be returned. The format of this structure depends on the channel type as specified by subchan_type (see below). If no meters are kept for this channel type, then subchan_ptr may be null.

logical_chan_ptr (Input)
is a pointer to a structure in which logical channel meters (those maintained for every logical channel) are to be returned. This structure has the following form:

```
dcl 1 logical_chan_meters based aligned,
      2 current_meters like lcte.meters,
      2 saved_meters like lcte.meters;
```

where:

current_meters
contains the current values of the logical channel meters. The format of lcte.meters is described by lct.incl.pl1.

saved_meters
contains the values of logical channel meters the last time a copy_meters order was issued.

parent_ptr (Input)
is a pointer to a structure in which multiplexer-specific meters maintained by the channel's parent multiplexer are to be returned. The format of this structure depends on the channel type as specified by parent_type (see below).

subchan_type (Output)
is the channel type of the channel. It may have any of the values described in multiplexer_types.incl.pl1.

parent_type (Output)
is the channel type of the channel's parent multiplexer. It may have any of the values described in multiplexer_types.incl.pl1.

Modes Operation

The modes operation is supported when the I/O switch is open. The recognized modes are listed below. Some modes have a complement indicated by the circumflex character (^) that turns the mode off (e.g., ^erkl). For these modes the complement is displayed with the mode. Normal defaults are indicated for those modes that are generally independent of terminal type. The modes string is processed from left to right. Thus, if two or more contradictory modes appear within the same modes string, the rightmost mode prevails.

8bit, ^8bit
causes input characters to be received without removing the 8th (high-order) bit, which is normally interpreted as a parity bit. This mode is valid for HSLA channels only. (Default is off.)

blk_xfer, ^blk_xfer
specifies that the user's terminal is capable of transmitting a block or "frame" of input all at once in response to a single keystroke. The system may not handle such input correctly unless blk_xfer mode is on and the set_framing_chars order has seen issued. (Default is off.)

breakall, ^breakall
: enables a mode in which all characters are assumed to be break characters, making each character available to the user process as soon as it is typed. This mode only affects get_chars operations. (Default is off.)

can, ^can
: performs standard canonicalization on input. (Default is on.)

can_type=overstrike, can_type=replace
: specifies the method to be used to convert an input string to canonical form. Canonicalization is only performed when the I/O switch is in "can" mode. (Default is can_type=overstrike.)

capo, ^capo
: outputs all lowercase letters in uppercase. If edited mode is on, uppercase letters are printed normally; if edited mode is off and capo mode is on, uppercase letters are preceded by an escape (\) character. (Default is off.)

crecho, ^crecho
: echoes a carriage return when a line feed is typed. This mode can only be used with terminals and line types capable of receiving and transmitting simultaneously.

ctl_char, ^ctl_char
: specifies that ASCII control characters that do not cause carriage or paper motion are to be accepted as input, except for the NUL character. If the mode is off, all such characters are discarded. (Default is off.)

default
: is a shorthand way of specifying erkl, can, ^rawi, ^rawo, ^wake_tbl, and esc. The settings for other modes are not affected.

echoplex, ^echoplex
: echoes all characters typed on the terminal. The same restriction applies as for crecho; it must also be possible to disable the terminal's local copy function.

edited, ^edited
: suppresses printing of characters for which there is no defined Multics equivalent on the device referenced. If edited mode is off, the 9-bit octal representation of the character is printed. (Default is off.)

erkl, ^erkl
: performs "erase" and "kill" processing on input. (Default is on.)

esc, ^esc
: enables escape processing (see "Typing Conventions" in Section 2) on all input read from the device. (Default is on.)

force
: specifies that if the modes string contains unrecognized or invalid modes, they are to be ignored and any valid modes are to be set. If force is not specified, invalid modes cause an error code to be returned, and no modes are set.

fulldpx, ^fulldpx
: allows the terminal to receive and transmit simultaneously. This mode should be explicitly enabled before enabling echoplex mode.

hndlquit, ^hndlquit
: echoes a newline character and performs a resetread of the associated stream when a quit signal is detected. (Default is on.)

iflow, ^iflow
: specifies that input flow control characters are to be recognized and/or sent to the terminal. The characters must be set before iflow mode can be turned on.

init
: sets all switch type modes off, sets line length to 50, and sets page length to zero.

lfecho, ^lfecho
: echoes and inserts a line feed in the user's input stream when a carriage return is typed. The same restriction applies as for crecho.

ll*n*, ^ll
: specifies the length in character positions of a terminal line. If an attempt is made to output a line longer than this length, the excess characters are placed on the next line. If ^ll is specified, line length checking is disabled. In this case, if a line of more than 255 column positions is output by a single call to iox_$put_chars, some extra white space may appear on the terminal.

no_outp, ^no_outp
: causes output characters to be sent to the terminal without the addition of parity bits. If this mode and rawo mode are on, any 8-bit pattern can be sent to the terminal. This mode is valid for HSLA channels only. (Default is off.)

oddp, ^oddp
: causes any parity generation that is done to the channel to assume odd parity. Otherwise, even parity is assumed for line types other than 2741 and 1050. This mode is valid for HSLA channels only. (Default is off.)

oflow, ^oflow
: specifies that output flow control characters are to be recognized when sent by the terminal. The characters and the protocol to be used must be set before oflow mode can be turned on.

pl*n*, ^pl
specifies the length in lines of a page. When an attempt is made to exceed this length, a warning message is printed. When the user types a formfeed or newline character (any break character), the output continues with the next page. The warning message is normally the string "EOP", but can be changed by means of the set_special control order. The string is displayed on a new line after *n* consecutive output lines are sent to the screen (including long lines which are folded as more than one output line). To have the end-of-page string displayed on the screen without scrolling lines off the top, *n* should be set to one less than the page length capability of the screen, unless the end-of-page string is a null string. In this case, output stops at the end of the last line of the page or screen. If ^pl is specified, end-of-page checking is disabled. (See description of scroll mode below.)

polite, ^polite
does not print output sent to the terminal while the user is typing input until the carriage is at the left margin, unless the user allows 30 seconds to pass without typing a newline. (Default is off.)

prefixnl, ^prefixnl
controls what happens when terminal output interrupts a partially complete input line. In prefixnl mode, a newline character is inserted in order to start the output at the left margin; in ^prefixnl mode, the output starts in the current column position. (Default is on.) Polite mode controls when input may be interrupted by output; prefixnl controls what happens when such an interruption occurs.

rawi, ^rawi
reads the data specified from the device directly without any conversion or processing. (Default is off.)

rawo, ^rawo
writes data to the device directly without any conversion or processing. (Default is off.)

red, ^red
sends red and black shifts to the terminal.

replay, ^replay
prints any partial input line that is interrupted by output at the conclusion of the output, and leaves the carriage in the same position as when the interruption occurred. (Default is off.)

scroll, ^scroll
specifies that end-of-page checking is performed in a manner suited to scrolling video terminals. If the mode is on, the end-of-page condition occurs only when a full page of output is displayed without intervening input lines. The mode is ignored whenever end-of-page checking is disabled. (Default is off.)

tabecho, ^tabecho
echoes the appropriate number of spaces when a horizontal tab is typed. The same restriction applies as for crecho.

tabs, ^tabs
inserts tabs in output in place of spaces when appropriate. If tabs mode is off, all tab characters are mapped into the appropriate number of spaces.

vertsp, ^vertsp
: performs the vertical tab and formfeed functions, and sends appropriate characters to the device. Otherwise, such characters are escaped. (Default is off.)

wake_tbl, ^wake_tbl
: causes input wakeups to occur only when specified wakeup characters are received. Wakeup characters are defined by the set_wakeup_table order. This mode cannot be set unless a wakeup table has been previously defined. *

Notes

The status code error_table_$action_not_performed is returned by the printer_on and printer_off control operations if the special characters table currently in effect indicates that this terminal cannot perform the printer_on or printer_off operation. The status code error_table_$no_table is returned by the get_input_translation, get_output_translation, get_input_conversion, get_output_conversion, and get_special control orders if the specified table does not exist. A code of zero is returned otherwise.

To assist the user in determining how to alter the tables described above, the following paragraphs provide a summary of the processing of input and output strings in ring 0.

INPUT PROCESSING

1. Translation
   The characters are translated from the terminal's code to ASCII, using the input_translation table. If there is no input_translation table, this step is omitted.

2. Canonicalization
   The input string is rearranged (if necessary) into canonical form as described in Section 2.

3. Editing
   Performs erase and kill processing as described in Section 2.

4. Break and escape processing
   The characters in the input string are looked up in the input_conversion table and treated accordingly. If a character is preceded by an escape character (as determined from the table) it is looked up in the input_escapes array in the special_chars table, and, if found, replaced by the corresponding character from the input_results array.

OUTPUT PROCESSING

1. Capitalization
   Lowercase letters are replaced by uppercase for terminals in "capo" mode; uppercase letters are prefixed by escape characters if appropriate.

2. Formatting
   The characters in the output string are looked up in the output_conversion table described above. Carriage-movement characters are replaced by sequences found in the special_chars table, followed by delay characters if so indicated by the delay table. Ribbon-shift characters are likewise replaced by appropriate sequences. Any character whose indicator in the output_conversion table is greater than 16 is replaced by the (indicator-16)th sequence in either the not_edited_escapes or edited_escapes array in the special_chars table.

3. Translation
   The result of step 2 is translated from ASCII to the terminal's code, using the output_translation table. If there is no output_translation table, this step is omitted.

Control Operations from Command Level

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

io_call control switch_name order_arg where:

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

2. order_arg
   can be any control order described above under "Control Operation" that can accept a null info_ptr, as well as read_status, write_status, terminal_info, and the following (which must be specified as shown):

   store_id id
      where id is the new answerback string.

   set_term_type type {-control_args}
      where type is the new terminal type and -control_args may be any of -initial_string (-istr), -modes, and -ignore_line_type.

   set_line_type line_type
      where line_type is the new line type.

   line_length N
      where N is the new line length.

The following control orders can be used as active functions:

[io_call control switch_name read_status]
   returns true if input is available; otherwise, false.

[io_call control switch_name write_status]
   returns true if output is pending; otherwise, false.

```
[io_call control switch_name terminal_info terminal_type]
    returns the current terminal type.

[io_call control switch_name terminal_info baud]
    returns the baud rate.

[io_call control switch_name terminal_info id]
    returns the terminal identifier (answerback).

[io_call control switch_name terminal_info line_type]
    returns the current line type.
```

Name: tty_printer_

The tty_printer_ I/O module performs stream I/O to a standard terminal (e.g., TN1200, ROSY, Diablo, VIP7760, or IBM3270 printer) to make it operate as a remote printer. The hardware options currently supported are defined by the control arguments described below.

The tty_printer_ I/O module can also be used to direct its stream I/O through the syn_ I/O module to another I/O switch (e.g., user_i/o or to a file switch through vfile_).

Entry points in this module are not called directly by users; rather, the module is accessed through the I/O system. It is normally attached through the remote_printer_ I/O module and all attach options are passed through remote_printer_ to tty_printer_.

Attach Description

tty_printer_ -control_args

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

-device STR
: attaches the switch as the device type specified by STR. STR is normally printer or teleprinter.

-auto_call N
: specifies the phone number, N, to be called via the automatic call unit on the specified communications channel.

-tty STR
: defines the target communications channel to be STR, where STR is an I/O switch name if the communications I/O module is syn_.

-comm STR
: uses the communications I/O module specified by STR. Normally, STR is either tty_ or syn_.

-physical_line_length N, -pll N
: specifies the physical line length, N, of the output device.

-terminal_type STR, -ttp STR
: STR specifies the terminal type whose conversion, translation, and special tables defined in the user or system terminal type table (TTT) are used to convert and translate input and output to and from the device. If not specified, the default terminal type is used.

-horizontal_tab, -htab
: specifies that horizontal tab characters are to be sent to the device.

-vtab
: specifies that vertical tab characters are to be sent to the device.

Open Operation

The tty_printer_ I/O module supports stream_input, stream_output, and stream_input_output opening modes.

Put Chars Operation

The put_chars entry passes the data directly to the communications I/O module without any conversion.

Get Chars/Get Line Operation

The get_chars and get_line entries pass the operation directly to the communications I/O module.

Control Operation

This I/O module passes all undefined control operations to the communications I/O module. In addition, it supports the control operations listed below. Unless otherwise specified, there are no input control structures.

select_device
: selects the device characteristics for which output is next directed. The device is the one associated with the I/O switch by the -device option at attachment. The input structure is of the form:

```
dcl device char(32);
```

runout
: transmits any data stored in the output buffer.

hangup_proc
: sets up a specified event call channel to be signalled over, and a procedure to be called, if the communications channel hangs up. The hangup_proc input structure has the following form:

```
dcl 1 hangup_proc aligned,
      2 entry     entry variable,
      2 datap     ptr,
      2 prior     fixed bin;
```

where:

entry
: is the entry to call when a hangup is detected.

datap
: is a pointer to data for the hangup procedure.

prior
: is the ipc_ event call priority to be associated with hangup notification.

reset
: sets the ^edited mode of output conversion and enables the tabs and vertsp modes if required by attachment options.

get_error_count
: returns the current count of errors detected since attachment. The input structure is of the form:

```
dcl error_count fixed bin;
```

hangup
: is used to hang up the device communications connection. This control operation is trapped if the communications I/O module is syn_, otherwise it is passed on.

Modes Operation

This I/O module passes all modes operations to the communications I/O module.

Notes

This I/O module is normally attached through a remote device I/O module (e.g. remote_printer_ or remote_teleprinter_.) Attachment to tty_printer_ is specified in the remote_device attach description by "-terminal tty_printer_" along with any attach options listed above. The -device attach option is supplied by the remote_device I/O module.

APPENDIX A

NAMES OF COMMUNICATIONS CHANNELS

The name of a communications channel is an encoding of the information describing the physical connection. Every such name is a string of 6 to 32 characters. The name is divided into components separated by "." characters; each component represents a level of multiplexing.

The first two components have a standard form, and describe a physical channel on an FNP. Multiplexed channels (i.e., subchannels of a concentrator whereby multiple terminals are supported on a single FNP channel) have additional components identifying the individual subchannels. The form of each component depends on the type of multiplexer involved.

The general form of the name of a physical channel is:

    F.ANSS

where:

F
: is a top-level multiplexer name. If this is an FNP, the name must be a, b, c, d, e, f, g, or h. Other system or user defined top-level multiplexers may have different naming conventions.

A
: is l for a channel of a low-speed line adapter (LSLA) or h for a channel of a high-speed line adapter (HSLA).

N
: is the number of the LSLA or HSLA on the specified FNP. It is in the range 0 to 5 for LSLAs or 0 to 2 for HSLAs.

SS
: is a 2-digit decimal number identifying a subchannel of the specified LSLA or HSLA.

T & D Channel

A channel called F.c000, where F is an FNP identifier, is a special virtual channel used by COLTS (Communications Online Test and Diagnostics System). It does not correspond to an actual physical channel on the FNP.

Examples

a.l003	FNP a, LSLA 0, subchannel 03
a.h219	FNP a, HSLA 2, subchannel 19
c.l411	FNP c, LSLA 4, subchannel 11

In the following examples, the physical channel b.h108 (i.e., FNP b, HSLA!1, subchannel 8) is assumed to be a concentrator whose subchannels are numbered sequentially from 0 to 15:

b.h108.00	subchannel 0 (first subchannel)
b.h108.03	subchannel 3
b.h108.15	subchannel 15 (last subchannel)

APPENDIX B

PRINTER MODES AND CONTROL ORDERS

The following are descriptions of the control operation and modes operation for the standard printer output module as supported by the remote_teleprinter_ and remote_printer_ I/O modules described in Section 6.

MODES

There are two mode types: binary and numerical. There is also a pseudo-mode, default, which sets all modes to their default values. A modes string is a string of mode keys separated by commas. The current value of a mode is changed when its mode key appears in the mode string. It is unchanged if the mode key is omitted from the mode string. Mode keys may appear in any order and if a key appears more than once, the last value is used.

Binary Modes

Each binary mode has two possible values. The mode is set if the mode key appears in the mode string. It is reset if the mode key begins with the "^" character. The binary mode keys are defined as follows:

1pg, ^1pg
: causes the output module to return to the caller when the end of the current page is reached (i.e., at the formfeed position for the next logical page). If there are unprocessed characters at this point, the code error_table_$request_pending is returned. The default is ^1pg.

ctl_char, ^ctl_char
: causes the output module to pass nonprinting characters to the device as is. Carriage movement characters (newline, formfeed, carriage return, backspace, and horizontal and vertical tab) are interpreted normally. The ASCII escape character (octal 033) is also transmitted directly, unless esc mode is enabled (see below). If ctl_char mode is disabled, the treatment of nonprinting characters is determined by the setting of non_edited mode. The default is ^ctl_char.

esc, ^esc
: enables searching for escape sequences in the input string, which enables slew to channel orders. The default is ^esc.

non_edited, ^non_edited
: causes the output module to print the applicable octal ASCII code preceded by a backslash (\) for nonprinting characters, and to use the nonedited output conversion table in the specified TTT for the remote device. The ^non_edited value causes any such characters to be omitted from the output. The setting of this mode is ignored when ctl_char is in effect. The default is ^non_edited.

noskip, ^noskip
: suppresses the automatic insertion of blank lines at the end of a logical page (i.e., it allows the printer to print over the perforations). It has the side effect of setting the logical page length to its default value. The default is ^noskip.

print, ^print
: specifies that processed characters from the input string are to be printed. The ^print value allows a string to be processed for output, sets page and line counts, and honors the 1pg and stopN modes, but without actually printing the processed characters. The default is print.

single, ^single
: specifies that any formfeed or vertical tab characters from the input string are to be converted to newline characters (i.e., it suppresses runaway paper feeding). The default is ^single.

truncate, ^truncate
: truncates the output if the line exceeds the line length. The ^truncate value allows the line to be wrapped onto the next line if it is too long. The default is ^truncate.

Numerical Modes

Numerical modes supply a value to be used during a put_chars operation. If the numerical portion of the mode cannot be converted to a binary number, a conversion error is signalled. The default values for numerical modes are set by the default pseudo-mode or by the reset control operation (see below). The numerical modes are defined as follows:

plN
: sets the logical page length to N lines. At the end of a logical page, the printer skips to the next formfeed position (unless noskip mode is set). The value of N must be greater than one, and can be greater than a physical page. The default value is physical page length minus lines per inch.

llN
: sets the logical line length to N characters. The value of N must be greater than the indentation (see below) and must not be greater than the physical line length of the device. The default value is the physical line length.

inN
: sets the indentation to N characters. The value of N must be 0 or a positive integer which is less than the logical line length. The default value!is!0.

stopN
: sets the output module to return to the caller every N pages even though the processing of the input string has not been completed. If there is unprocessed input remaining, a code of error_table_$request_pending is returned. A value of 0 means do not return until all input is processed. The counter of how many pages to process before returning is reset when a new value is given. The default value is 0.

CONTROL ORDERS

The control orders for the printer output module sometimes take an info pointer argument. Each of these is identified in the following descriptions. Each info pointer describes a structure which contains additional data or provides a place for data to be written. All structures used for control orders are contained in the prt_order_info.incl.pl1 include file. The control orders are defined as follows:

channel_stops
: sets the channel stop data used for slew to channel control sequences during a put_chars operation. The info pointer defines the channel_stops input array as found in the prt_order_info include file. Array element N defines the stops for line number N. Bit M of an array element defines a stop for channel M. The initial value is no stops defined. Once defined, the stops remain in effect until the next channel_stops control operation.

end_of_page
: advances the paper to the bottom of the current page, one line below the point where page labels are printed. If page labels are set the label is printed. The info pointer is not used and may be null.

get_count
: returns accounting information. The info pointer defines the counts output structure as found in the prt_order_info include file. The page and line counts are reset by the reset control operation.

get_error_count
: returns the error count since the output module was attached. The info pointer defines the output variable ret_error_count as found in the prt_order_info include file.

get_position
: returns the position data defined by the position data structure in the prt_order_info include file. The data resembles that of the get_count control operation, but the structure adds the total characters printed since the last reset to allow the caller to start the next put_chars operation at the following character when the module returns due to 1pg or stopN mode. The data structure is also used for the set_position operation (see below).

inside_page
: advances the paper to the formfeed position of the next inside page. An inside page is a top page when the listing is folded correctly. Separator bars for the head sheet are printed over the perforations at the bottom of an inside page. The info pointer is not used and may be null.

outside_page
: advances the paper to the formfeed position of the next outside page. An outside page is a bottom page when the listing is folded correctly. The info pointer is not used and may be null.

page_labels
: sets the top and bottom page labels to be printed for each logical page. The info pointer may be null to reset page labels to blank. Otherwise, the info pointer defines the page_labels input structure as found in the prt_order_info include file.

paper_info
sets the physical characteristics of the paper in the printer. The info pointer defines the paper_info input structure as found in the prt_order_info include file. Once set, the paper_info remains in effect until the next paper_info control operation. If the printer has a software loadable VFC image, a new image is loaded and the printer placed out of synchronization for the operator to align the paper. Otherwise, the code error_table_$no_operation is returned so the caller can request the operator to load the appropriate VFU tape and set the required lines per inch switch to complete the operation. The defaults are: page length, 66; line length, 136; lines per inch, 6.

reset
resets the output module to its default state: default modes, no page labels, line count = 0, page count = 1, and total chars = 0. The info pointer is not used and may be null.

resetwrite
cancels any data buffered for output. It is used to clear the output module after an error so the paper can be resynchronized. The info pointer is not used and may be null.

runout
causes all buffered data to be output before returning to the caller. It is used to synchronize the program with the actual device. The info pointer is not used and may be null.

set_position
sets the internal counters in the output module. The info pointer defines the position_data input structure as found in the prt_order_info include file. This is the reverse of the get_position control operation. It is used to start the accounting data at the correct point when restarting an I/O daemon request in the middle.

MULTICS PROGRAMMERS' MANUAL COMMUNICATIONS INPUT/OUTPUT ADDENDUM A

SUBJECT

Additions and Changes to the Manual

SPECIAL INSTRUCTIONS

This manual is one of six manuals that constitute the *Multics Programmers' Manual (MPM)*.

Order Number	*Title*
AG91	*Reference Guide*
AG92	*Commands and Active Functions*
AG93	*Subroutines*
AK92	*Subsystem Writer's Guide*
AX49	*Peripheral Input/Output*
CC92	*Communications Input/Output*

This is the first addendum to CC92, Rev. 1, dated August 1981.

Insert the attached pages into the manual according to the collating instructions on the back of this cover.

Throughout the manual, change bars in the margins indicate technical additions and changes; asterisks denote deletions. These changes will be incorporated into the next revision of this manual.

Note:
Insert this cover behind the manual cover to indicate that the manual has been updated with this Addendum.

SOFTWARE SUPPORTED

Multics Software Release 10.0

ORDER NUMBER

CC92-01A

July 1982

34844
1682
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update the manual, remove old pages and insert new pages as follows:

Remove	Insert
iii, iv	iii, iv
v, blank	v, blank
3-5, 3-6	3-5, 3-6
4-1, 4-2	4-1, 4-2 4-2.1, 4-2.2 4-2.3, blank
4-11, 4-12	4-11, 4-12
6-7, 6-8	6-7, 6-8
6-27, 6-28	6-27, 6-28
6-47 through 6-52	6-47, 6-48 6-48.1, blank 6-49 through 6-52
6-63 through 6-66	6-63 through 6-66
6-69 through 6-72	6-69 through 6-72
B-1 through B-4	B-1 through B-4
i-1 through i-4	i-1 through i-5, blank

File No.: 1U63, 1L63

7/82

CC92A

INDEX

0

2-10

@ 2-10

\ 2-11

A

answerback 3-2

answerback entry
  example of 3-21

answerback table 3-18

answerback table entry
  answerback statement 3-18
  type statement 3-19

ASCII character set 2-1
  control characters 2-1, 2-3
  null 2-4
  printing graphics 2-1
    special characters 2-1
  reserved characters 2-4

attachment 1-1

autocall
  dial_out command

B

baud rate 2-17, 3-2

bisync 6-2
  abort order 6-4
  end_write_mode order 6-5
  event_info order 6-4
  get_bid_limit order 6-4
  get_bsc_modes order 6-4
  get_chars order 6-5
  get_multi_record_mode order 6-5
  get_size order 6-4
  hangup order 6-4
  hangup_proc order 6-6
  read_status order 6-4
  resetread order 6-4
  resetwrite order 6-4

bisync (cont)
  runout order 6-4
  send_nontransparent_msg order 6-5
  set_bid_limit order 6-4
  set_bsc_modes order 6-4
  set_multi_record_mode order 6-5
  set_size order 6-4
  write_status order 6-4

block transfer 2-20

C

canonical form 2-4
  canonicalization 2-6, 2-12
    column assignment 2-6
    overstrikes 2-5, 2-7, 2-13
    replacement 2-8
    tab 2-13

carriage motion 2-5, 2-16

Changing Terminal Type Definitions
    3-2

channel 1-1, 3-1

channel definition table 3-2

character conversion 1-2
  set_tty 4-9

communications channels
  binary synchronous
    bisync 6-2
  naming A-1

continuation lines 2-17

control orders B-1, B-3

conversion table 3-15
  example of 3-21
  input 3-8
  output 3-8

cv_ttf command 3-2, 3-3, 4-2

D

default_types statement
  example of 3-20

delay table 2-17

delays 2-17, 6-53
  set_tty 4-9

dial facility 1-1

dial_out command 4-2.1

display_ttt command 3-1, 3-3, 4-3

E

end-of-page processing 2-18

erase character
  designation 2-10

erase processing 2-6, 2-10, 2-13
  escape sequences 2-11, 2-15
  overstrikes 2-10, 2-14

escape conventions 2-11, 2-18

escape processing 2-6

escape sequence 2-11, 2-17

escape sequence character 2-11

F

flow control 2-20
  input 2-20
  output 2-20

FNP
  see Front-End Network Processor

framing characters 6-60

Front-End Network Processor 1-1

function key table 3-15
  example of 3-15

G

g115_ 6-7
  end_write_mode order 6-8, 6-9
  hangup_proc order 6-8
  reset order 6-8
  runout order 6-8
  select_device order 6-8

g115_ orders
  see tty_ orders

H

hasp_host_ 6-10
  end_write_mode order 6-13
  hangup_proc order 6-14

hasp_host_ (cont)
  no_signon_record order 6-14
  read_status order 6-14
  reset order 6-14
  resetread order 6-14
  resetwrite order 6-14
  runout order 6-13
  select_device order 6-14
  SIGNON processing 6-15
    no_signon_record order 6-16
    signon_record order 6-15
  signon_record order 6-14

hasp_workstation_ 6-18
  carriage control specifications 6-24
  character set specification 6-23
  end_write_mode order 6-22
  hangup_proc order 6-22
  read_status order 6-22
  reset order 6-23
  resetread order 6-22
  resetwrite order 6-22
  runout order 6-22
  select_device order 6-23

I

I/O modules
  bisync_ 6-2
  g115_ 6-7
  hasp_host_ 6-10
  hasp_workstation_ 6-18
  ibm2780_ 6-25
  ibm3270_ 6-28
  ibm3780_ 6-36
  prtdim_ B-1
  remote_input_ 6-39
  remote_printer_ 6-41
  remote_punch_ 6-43
  remote_teleprinter_ 6-46
  tty_ 6-48
  tty_printer_ 6-74

ibm2780_ 6-25
  select_device order 6-27
  set_bsc_modes order 6-27
  set_multi_record_mode order 6-27

ibm3270_ 6-28
  event_info order 6-29
  general_poll order 6-29
  get_input_message_size order 6-35
  poll order 6-29
  read order 6-30
  set_input_message_size order 6-35
  stop_general_poll order 6-29
  write order 6-30

ibm3270_ orders
  see tty_ orders

ibm3780_ 6-36
  select_device order 6-38
  set_bsc_modes order 6-38
  set_multi_record_mode order 6-38

input conversion table 3-8

K

kill character
  designation 2-10

kill processing 2-6, 2-10, 2-13
  overstrikes 2-10

L

l6_ftf command 4-4

line type 3-1, 3-2

login command
  -terminal_type 3-2

M

modes
  binary B-1
  numerical B-2

O

output conversion table 2-17, 3-8

P

preaccess command entry
  preaccess_command statement 3-19
  type statement 3-19

printer modes B-1

print_terminal_types (ptt) command
    3-1, 4-6

print_ttt_path 4-7

prtdim_
  binary modes B-1
  channel_stops order B-3
  end_of_page order B-3
  get_count order B-3
  get_error_count order B-3
  get_position order B-3
  inside_page order B-3
  numerical modes B-2
  outside_page order B-3
  page_labels order B-3
  paper_info order B-4
  reset order B-4
  resetwrite order B-4
  runout order B-4
  set_position order B-4

ptt
  see print_terminal_types command

R

raw mode
  input 1-2
  output 1-2

remote_input_ 6-39
  get_count order 6-40
  reset order 6-40

remote_printer_ 6-41

remote_punch_ 6-43

remote_teleprinter_ 6-46

S

setting terminal types 3-2

set_ttt_path command 3-1, 4-8

set_tty (stty) command 3-2, 4-9

special characters table 2-16, 3-8,
    3-16
  example of 3-21
  special_table statement 3-16

special characters table entry
  backspace statement 3-17
  black_shift statement 3-17
  carriage_return statement 3-16
  edited_output_escapes statement
    3-17
  end_of_page statement 3-17
  form_feed statement 3-17
  input_escapes statement 3-18
  new_line statement 3-16
  output_escapes statement 3-17
  printer_off statement 3-17
  printer_on statement 3-17
  red_shift statement 3-17
  tab statement 3-17
  vertical_tab statement 3-17

stty
  see set_tty command

T

tab 2-6

terminal 1-1

terminal type 3-1
  changing 3-2
  preaccess commands 3-19
  setting 3-2

terminal type entry 3-5
  additional_info statement 3-6
  bauds statement 3-6
  bps statement 3-6
  buffer_size statement 3-10
  cps statement 3-6

terminal type entry (cont)
  delay statement 3-7
  erase statement 3-7
  examples of 3-20
  framing_chars statement 3-9
  function_keys statement 3-5
  global statement 3-14
  initial_string statement 3-5
  input_conversion statement 3-8
  input_resume statement 3-9
  input_suspend statement 3-9
  input_translation statement 3-9
  keyboard_addressing statement 3-8
  kill statement 3-8
  line_delimiter statement 3-8
  line_types statement 3-7
  modes statement 3-5
  output_acknowledge statment 3-10
  output_end_of_block statement 3-10
  output_resume_statement 3-10
  output_suspend statement 3-9
  set_input_conversion order 3-15
  set_output_conversion order 3-15
  special statement 3-8
  terminal_type statement 3-5
  video table 3-10

terminal type file 2-17, 3-2, 3-4
  character specifications 3-4
  cv_ttf 4-2
  default_types statement 3-18
  display_ttt 4-3
  global statements 3-14

terminal type table 3-1
  cv_ttf 4-2
  default TTT 3-3
  display_ttt 4-3
  print_terminal_types 4-6
  set_ttt_path 4-8
  ttt_info_ 5-2

terminal types
  print_terminal_types 4-6
  set_tty 4-9

translation table 3-1, 3-9, 3-15
  example of 3-21

ttt_info_ 3-1, 5-2
  ttt_info_$additional_info 5-6
  ttt_info_$decode_answerback 5-8
  ttt_info_$decode_type 5-9
  ttt_info_$dialup_flags 5-7
  ttt_info_$encode_type 5-8
  ttt_info_$function_key_data 5-14
  ttt_info_$initial_string 5-6
  ttt_info_$modes 5-5
  ttt_info_$preaccess_type 5-5
  ttt_info_$terminal_data 5-2
  ttt_info_$video_info 5-10

tty_ 6-48
  abort order 6-50
  accept_printer_off order 6-52
  answerback
    terminal_info order 6-65
    wru order 6-51
  break and escape processing 6-71
  buffering 6-49
  canonicalization 6-71

tty_ (cont)
  capitalization 6-72
  copy_meters order 6-66
  editing 6-49, 6-71
  formatting 6-72
  get_delay order 6-54
  get_editing_chars order 6-54
  get_event order 6-66
  get_framing_chars order 6-60
  get_ifc_info order 6-63
  get_input_conversion order 6-56
  get_input_translation order 6-56
  get_meters order 6-66
  get_ofc_info order 6-63
  get_output_conversion order 6-56
  get_output_translation order 6-56
  get_special order 6-59
  hangup order 6-50
  input_flow_control_chars order 6-62
  interrupt order 6-50
  line type
    terminal_info order 6-65
  line_type
    set_line_type order 6-51
  listen order 6-51
  modes operation 6-67
  output delays
    get_delay order 6-54
    set_delay order 6-53
  output_flow_control_chars order
      6-62
  printer_off order 6-51
  printer_on order 6-51
  quit_disable order 6-64
  quit_enable order 6-64
  read_status order 6-64
  refuse_printer_off order 6-52
  resetread order 6-50
  resetwrite order 6-50
  send_initial_string order 6-65
  set_default_modes order 6-65
  set_delay order 6-53
  set_editing_chars order 6-54
  set_event order 6-65
  set_framing_chars order 6-60
  set_input_conversion order 6-55
  set_input_translation order 6-55
  set_line_type order 6-51
  set_output_conversion order 6-56
  set_output_translation order 6-55
  set_special order 6-57
  set_term_type order 6-60
  set_wakeup_table order 6-61
  start order 6-64
  start_xmit_hd order 6-51
  stop_xmit_hd order 6-51
  store_id order 6-64
  terminal type
    terminal_info order 6-65
  terminal_info order 6-65
  translation 6-71, 6-72
  write_status order 6-64
  wru order 6-51

tty_ modes 6-67
  blk_xfer
    framing characters 4-10
  can 2-5
  ctl_char 2-7
  edited 2-3, 2-17, 3-17
  erkl 2-5

tty_ modes (cont)
  esc 2-5, 2-10
  rawi 1-2, 2-5
  rawo 1-2
  wake_tbl 6-71

tty_ orders 6-50

tty_printer_ 6-74
  orders
    get_error_count 6-76
    hangup 6-76
    hangup_proc 6-75
    reset 6-76
    runout 6-75
    select_device 6-75

tty_printer_ orders
  see tty_printer_

V

video table 3-10
  control sequences 3-11

video table
  function simulation 3-13

video table
  global statement 3-14
  required functions 3-14
  statements 3-12

W

white space 2-10

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	MULTICS PROGRAMMERS' MANUAL COMMUNICATIONS INPUT/OUTPUT ADDENDUM A	ORDER NO.	CC92-01A
		DATED	JULY 1982

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 ________________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

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

Honeywell

Honeywell

Honeywell Information Systems
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

32496, 5C981, Printed in U.S.A.

CC92-01