To: Distribution

From: Robert S. Coren & Jerry Stern

Date: 03/22/77

Subject: Multics Terminal Types

INTRODUCTION

This document describes a proposal to reimplement the handling of terminal types by MCS in such a way as to allow each site to define its own set of terminal types. The principal reason for doing this is that, with the wide (and increasing) variety of terminals available, the rather limited set defined by the present Multics software is becoming increasingly inadequate. To expand the list of statically-defined types, and their associated ring 0 tables, to include all known and conceivable terminals would be a waste of space even if it were possible. Users can, of course, redefine their own terminals on an individual basis by substituting the appropriate tables (as described in MTB 290), but it is unreasonable to require a large community of users, all using a particular terminal that does not happen to be precisely described by any of the system-supplied definitions, to take this special action in order to get their terminals to run properly. By means of a replaceable table, generated from an ASCII file that can be edited by a system administrator, a site can describe with whatever precision it desires the terminals it expects to be most commonly connected to its system.

In order to explain how this can be done, it seems advisable first to define what is meant by a "terminal type" from the point of view of MCS (something that has never been done before), and to describe the presently-implemented method of determining and acting on a given terminal's type.

DEFINITION_OF_"TERMINAL_TYPE"

The concept of "terminal type" is used to distinguish among terminals on the basis of their physical characteristics and behavior. In particular, the following attributes are components of

a terminal's type:

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

A terminal type is to be distinguished from a "line type," which defines the communications protocol used to control a terminal or related device. In MCS, the terminal type associated with a communications channel is used by the ring 0 software, whereas the line type is primarily of interest to the FNP.

From the point of view of the ring 0 software, a terminal type is specified by a set of tables (used by tty_read and tty_write when processing input and output respectively) and a few other attributes. These items are summarized below. The tables are described in MTB 290; their use is described in more detail in MTBs 234 and 262. Most of the tables and attributes can be changed by a user on an individual basis; the terminal type is a way of specifying a complete set of consistent attributes.

Translation Tables

Input and output translation tables are used to translate between ASCII and the terminal's character code. These tables may be absent for an ASCII terminal.

Input_Conversion_Table

The input conversion table is used to identify certain special characters on input, such as escape characters.

Output_Conversion_Table

The output conversion table is used in conjunction with the special characters table (see below) to identify carriage control characters and characters that are to be replaced by control sequences or escape sequences (escape sequences are used to fill gaps in the terminal's character set).

Special_Characters_Table

The special characters table specifies control sequences for carriage movement, ribbon shift, and the disabling and enabling of local copy ("printer off" and "printer on"); in addition, it contains output escape sequences and defines the results of input escape sequences.

Delay_Tables

A delay table specifies the number of "delay" (NUL) characters required to allow the terminal to perform various kinds of carriage motion. Clearly these numbers depend on the speed of the terminal; accordingly, several different delay tables may be associated with a single terminal type, one for each of several speeds.

Other_Attributes

INITIAL MODES

A string suitable for passing to iox_$modes is provided to specify the minimal set of modes suitable to the terminal. Page length (if any) and line length are included here; some other modes related to physical characteristics of a terminal are

lfecho, crecho, tabecho, tabs, and vertsp.

HORIZONTAL TAB SETTING

For some terminals, horizontal tab stops are settable by software, by sending a control sequence to the terminal. For such terminals, a string is supplied that may be used to set tab stops at the Multics standard tab positions (columns 1, 11, 21, etc.).

EDITING CHARACTERS

The editing characters are those recognized by tty_read as "erase" and "kill" characters. The selection of these characters is actually a software consideration rather than an attribute of the terminal (except in the case of a terminal that lacks either of the standard editing characters); however, an initial specification of these characters is included for convenience as part of the specification of a terminal type.

KEYBOARD LOCKING AND UNLOCKING

Some terminals lock and unlock their keyboards in response to certain control sequences. These control sequences are defined on the basis of line type in the FNP software. If the terminal's line type is "ASCII" (normal ASCII asynchronous protocol) the use of this feature is controlled by the terminal type; for example, the Teletype Model 37 has this feature, while the TermiNet 300 does not. For IBM line types (2741 and 1050) the locking and unlocking of the keyboard is essential, and cannot be disabled; for other line types the feature is not supported.

PRESENT IMPLEMENTATION OF TERMINAL TYPES

Ten terminal types are recognized by MCS at present, and several programs contain assumptions about possible types; the addition of other types is therefore difficult without major restructuring such as is proposed in this MTB. Most of the information needed to specify the terminal types is kept in a ring 0 segment named tty_tables, which is generated from a mexp source

segment. These types are known in ring 0 by numbers; an include file used in the user ring associates names with these numbers. The terminal type number is an index into an array of structures at the base of tty_tables that contain the offsets of the tables used by the various types. In particular, each structure in the array contains the offsets of the input and output translation tables, the input and output conversion tables, the special characters table, and delay tables to be used at 10, 15, 30, and 120 characters per second. A 0 offset means that the corresponding table does not exist for the particular terminal type. When a terminal's type is determined initially when the terminal dials up, or is changed by means of a "set_type" order, pointers to these tables are derived from the offsets described above and stored in the terminal's ring 0 control block. These pointers may be replaced individually with pointers supplied by the user of the terminal; since, however, these are pointers in the user's address space, the pointers to the default tables in tty_tables must be restored whenever the terminal is assigned to another process.

The initial modes and the tab-setting strings for each terminal type are kept in a user-ring data base called the initial modes table, which is indexed (like tty_tables) by terminal type number. A modes call using the string supplied in the initial modes table is made whenever the terminal type is set.

Editing characters default for all terminal types to "#" and "@" for erase and kill respectively. They may be replaced by individual users.

The keyboard-locking feature is enabled for terminal type 3 (TTY37) and disabled for all other ASCII terminal types.

The initial terminal type is assigned on the basis of an "experiment" conducted by the answering service, using the baud rate, answerback, line type, and information stored in the CDT.

OVERVIEW OF PROPOSED CHANGES

In order to permit the definition of terminal types on a per installation basis, a new administrative data base will be created. The system administrator will maintain an ASCII source file for this data base called the Terminal Type File (TTF). The TTF will be converted to a binary data base called the Terminal Type Table (TTT) by a reduction-compiler translator. All terminal type information will reside in the TTT, a ring 4 segment ac-

cessible to the answering service and users alike. Each terminal type will be identified by an ASCII name.

Whenever a terminal type is set for a specified terminal, all of the needed terminal type data will be copied into ring 0. Normally, such data will be extracted from the TTT. However, this is not strictly necessary. The TTT merely represents a user ring convention for associating a name with a collection of terminal type data. This convention will be observed by the user ring tty_ I/O module. However, it will be possible for a user to bypass this convention, if desired. Thus, a user can manufacture his own terminal type data and present it to ring 0. Ring 0 requires only that the data be self-consistent. As an aid to the user ring, ring 0 will remember the terminal type name, but will not depend on it in any manner.

The ring 0 data bases related to terminal management will be reorganized to accommodate the new method of defining terminal types (and also to make certain general improvements). Currently, the segment called tty_tables is used to hold static terminal type data. Under the new scheme, tty_tables will be used to store terminal type data received from the user ring. Since it is expected that at any given site many users will share the same terminal types, space usage in tty_tables can be minimized by sharing various items of terminal type data. The wired segment called tty_buf is currently used to store terminal I/O buffers as well as two different data structures for each terminal. Since these data structures will have to be enlarged, it is sensible to separate information that needs to be wired (for reference at interrupt time) from information that need not be wired. Therefore, a new unwired segment called tty_data will be created to store unwired terminal data. Wired terminal data will continue to be stored in tty_buf.

One of the principal motivations for providing installation-defined terminal types is so that the answering service can correctly determine terminal characteristics at dialup time. Therefore, the answering service will be modified to compute a terminal type at dialup based on factors such as the terminal baud rate, the line type, the answerback (if any), and an optional default terminal type specified in the CDT. The way in which these factors are combined to select a terminal type is controlled by the system administrator.

IMPLEMENTATION

User_Ring

User ring changes required to support installation-defined terminal types include the addition of new orders to tty_, the

handling of obsolete orders by tty_, and modifications to the set_tty command. Also, the user_info_ subroutine and the user active function must be changed.

The new orders for tty_ include set_terminal_type, terminal_info, and set_tabs. The set_terminal_type order replaces the current set_type order. Given a terminal type name, it will extract the relevant data from the TTT and pass this data into ring 0. The caller specifies whether tabs and/or modes are to be set according to the default values for the terminal type. Normally, the terminal type for a given terminal is required to be compatible with the line type. However, the set_terminal_type order allows this restriction to be overridden. The terminal_info order is similar to the current info order. However, terminal_info returns a terminal type name (rather than a number) and will also indicate the line type. The set_tabs order transmits a tab-setting string to a given terminal. If no info structure is supplied, then the default tab string for the terminal type (if any) is gotten from the TTT. Otherwise, a tab-setting string is described by the info structure. Declarations for the info structures used by the three new orders are given in the Appendix.

The new scheme for defining terminal types by name will obsolete the terminal type numbers now used. This will affect the set_type and info orders to tty_. Compatibility considerations require that these two orders continue to work, at least for a moderate period of time. Therefore, the preparer of the TTF will specify an "old" terminal type number for each terminal type. This will permit tty_ to map a set_type order into a set_terminal_type order. Furthermore, set_terminal_type will pass the old type number into ring 0 along with the other terminal type data. This will permit ring 0, and hence tty_, to continue to support the info order. Of course, new terminal types will undoubtedly be defined for which no old counterpart exists. In this case, the TTT will specify an old type number of -1. User programs that call the info order will not recognize the -1, just as they would not recognize any new terminal type number.

The set_tty command must be changed to take advantage of the three new orders to tty_. This should actually simplify set_tty considerably. It is proposed that the interpretation of the -reset control argument be modified. Currently, -reset turns off all modes that are not turned on in the default mode string for the terminal type. (The default mode string normally specifies only positive modes.) This has the undesirable effect of turning off modes that are irrelevant to the terminal type. For example, the presence or absence of replay mode is of no consequence to most terminal types. Therefore, the default mode string for a terminal type should specify all required modes, both positive and negative. Then, "set_tty -reset" will simply set the default modes and leave other modes unchanged.

The tty_data entry point to the user_info_ subroutine returns a terminal type number and therefore will become obsolete. A new entry point called terminal_data will be provided as a replacement and will return a terminal type name. Of course, the old entry point must be retained for compatibility. Hence, user_info_$tty_data will return the "old" terminal type number from the TTT. Since user_info_ obtains its information from the PIT, both the terminal type name and the old terminal type number must be stored in the PIT by the answering service. This requires adding a new field, the terminal type name, to the PIT. The user active function must be changed to call user_info_$terminal_data rather than user_info_$tty_data to handle the "term_type" keyword.

As mentioned earlier, it is possible for a user program to create its own terminal type data and to pass this data to ring 0. In general, however, this would require at least a moderate amount of programming effort. To reduce this effort, it is intended that a user be able to employ the TTF translator to produce his own private TTT. This provides a relatively simple means for the knowledgeable user to create the necessary terminal type data. This still requires, however, that the user deal directly with the ring 0 interface. This inconvenience could be eliminated by the addition of another order to tty_ called set_terminal_type_table. As the name implies, this order would instruct tty_ to reference the user's TTT rather than the system TTT. Thus, setting a non-standard terminal type becomes as easy as setting a standard terminal type. Although such a feature could be readily supplied, it is not known to be worthwhile. It is expected that, in general, a site will supply a TTT that can accommodate all of its users. Therefore, the set_terminal_type_table order may not be implemented initially.

Ring_0

Changes to ring 0 for installation-defined terminal types include the addition of new orders to ring 0 MCS, the handling of obsolete orders, and the management of terminal type data supplied by the user ring. Also, per-channel data structures will be reorganized to separate wired and unwired data.

Two new orders for ring 0 MCS are called set_terminal_data and terminal_info. The set_terminal_data order accepts an info structure (see Appendix for declaration) that defines the various items of terminal type data. Also included is the old terminal type number. The user ring tty_ I/O module calls the set_terminal_data order when it receives a set_terminal_type order. The terminal_info order, as implemented by ring 0, is equivalent to the user ring terminal_info order.

The set_type and info orders now supported by ring 0 MCS will become obsolete. As described earlier, ring 0 can continue to support the info order because the set_terminal_data order obtains the old terminal type number. However, the set_type order can no longer be supported at the ring 0 interface. Any user programs that call ring 0 directly must be changed to use set_terminal_data rather than set_type.

The tty_tables segment will become a dynamic data base used to store translation tables, conversion tables, special character tables, and delay tables. These tables are set collectively by the set_terminal_data order and individually by several other existing orders. Since orders for setting individual tables allow the caller to specify that the default table for the terminal type be used, ring 0 MCS will always remember the default tables for a terminal type (as supplied by a set_terminal_data order) even after an individual table has been replaced. Currently, the use of a non-standard table requires that the user provide permanent storage for the table in the user ring. This will no longer be necessary.

A new program will be provided to manage tty_tables. This program will be responsible for the allocation and deallocation of table storage space. Each time a new table is about to be added, a search will be made to discover if an identical table is already present in tty_tables. If so, the new table need not be added. However, a reference count must be maintained in order to properly interpret subsequent requests for table deletion.

Two data structures called the FCTL and the CTL are now maintained for each terminal channel. These structures must be expanded to contain new pieces of terminal type information. Both CTLs and FCTLs are stored in tty_buf, a wired segment. Since much of the information in the CTL need not be wired, this information will be moved to an unwired segment called tty_data. To facilitate the reorganization, CTLs and FCTLs will be replaced by new data structures called TCBs (terminal control blocks) and WTCBs (wired terminal control blocks).

Answering_Service

Answering service changes required for installation-defined terminal types include a new method for determining terminal types at dialup time, use of a new order to set terminal types, and a TTT installation mechanism. Also of interest is the disposition of terminal types following a "new_proc" or "logout -hold".

The determination of a terminal type at dialup time will depend upon a combination of factors. As is presently the case, the answerback from a terminal, if any, can be used to ascertain

the terminal type. However, before the answerback can be read, an initial terminal type must be set. Currently, each CDT entry contains an optional default terminal type number. This number must be replaced by an ASCII name. If a default terminal type is specified in the CDT entry for a terminal, then that type is used for the initial terminal type. If no default type is specified in the CDT, then the answering service will examine a new data base contained in the TTT. This data base consists of an ordered list of triples, each containing a baud rate, line type, and terminal type. A special symbol is used to indicate a "wild card" baud rate or line type. The list is scanned until a match is found for the baud rate and line type of the given terminal. The associated terminal type becomes the initial terminal type.

Having set the initial terminal type, the answering service then checks a new flag in the CDT entry for the terminal indicating whether an attempt should be made to read the answerback. If so, the answerback is read and decoded according to another new data base contained in the TTT. This data base describes how to recognize answerbacks that indicate specific terminal types and also how to separate the terminal ID. If the answerback indicates a different type from the initial type, then the new terminal type is set.

In order to set terminal types, the answering service must make use of a new order. The answering service does not use tty_, but rather uses ring 0 MCS directly. Therefore, the answering service must use the set_terminal_data order. The answering service will also set tabs and modes whenever it sets a terminal type. For tab setting, it will share a common subroutine with tty_.

A mechanism for installing a new TTT must be added to the answering service. This will be handled in a manner similar to that of other tables. A new TTT must be checked to ensure that all default terminal types mentioned in the CDT are defined in the TTT. Conversely, a new CDT must satisfy the same check.

One aspect of installing a new TTT is different from that of most other administrative tables. The TTT is not only an answering service data base, but also a user data base. The deletion of an old TTT could adversely affect user processes. Therefore, when a new TTT is installed, the old copy will be renamed and not deleted until the next answering service initialization.

There is some question as to what effect a "new_proc" or "logout -hold" should have on terminal attributes. The current implementation is peculiar in that certain terminal characteristics, e.g. modes, are retained whereas other characteristics, e.g. translation tables, are reset to the default for the current terminal type. The terminal type itself is not changed. This odd behavior is clearly undesirable and no longer made necessary

by implementation considerations.

In the case of a new_proc, it is proposed that all terminal characteristics be retained since this is most often the desired result. Occasionally, a user may get into trouble by improperly adjusting certain terminal attributes and would like new_proc to rescue him. It will not.

The case of logout -hold is somewhat less clear-cut. This operation may be used by a person who is simply changing projects and wishes to retain the same terminal characteristics. On the other hand, the terminal may be passing from one user to another. The second user may or may not wish to retain the first user's terminal characteristics. As a matter of clarity (and perhaps security), it seems sensible that each new login should produce a predictable result unaffected by the actions of previous users. Furthermore, the new terminal type scheme should increase the likelihood that the terminal type selected by the answering service is the one the user wants. Hence, there seems little reason to retain terminal characteristics following a logout -hold.

System_Administrator

The system administrator will have the responsibility of maintaining the TTF. A standard TTF will be distributed with each system release and will define all commonly used terminal types. Although the system administrator will have the freedom to change terminal type names in the standard TTF, to do so would probably cause compatibility problems. User programs already depend on recognizing certain standard terminal type names. Therefore, these names should be retained.

As described previously, installation-time checks will be made to ensure the mutual consistency of the CDT and the TTT. From the administrator's point of view, however, it is somewhat inconvenient to discover an error at installation time. It would be more convenient if such errors could be detected at compilation time. Therefore, cv_cmf, the CMF to CDT translator, will be made to check that each terminal type specified in the CMF is actually defined in the installed TTT. This is the only way to verify a terminal type name. A warning will be issued if an unknown terminal type is found. Having the TTF to TTT translator perform the reverse check seems unnecessary.

Installation_Plan

The new terminal type scheme requires a combination of hardcore, answering service, and system library changes. Fortunately, however, it is possible to accomplish the necessary changes without requiring combined hardcore and online installa-

tions. An installation plan specifying major steps is given below.

1. Add interim versions of the new set_terminal_data and terminal_info orders to ring 0 MCS. The interim set_terminal_data will ignore all information except the old terminal type number and will behave exactly like the set_type order. The interim terminal_info will convert a terminal type number to a terminal type name.

2. Create a TTF containing information equivalent to that of the current tty_tables. Compile this to obtain a TTT and install it.

3. Use the new cv_cmf to compile the current CMF into a new CDT. The new CDT will contain new fields (terminal_type_name and read_answerback_flag).

4. In a special session, install the new answering service. Then use the new answering service to install the new CDT.

5. Install the new ring 0 MCS.

6. Install online changes.

The following is draft documentation for the System Administrator's Manual describing the syntax of the terminal type file. Several references are made to the description of some orders to the tty_ I/O module in the MPM Subsystem Writer's Guide; this version of the SWG has not been published, but the same information is presented in MTB 290.

SYNTAX OF THE TTF

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 PL/I-style 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.

Terminal_Type_Entry

The entry for each terminal type consists of a terminal_type statement naming the terminal type, followed by statements describing the attributes of the 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 16 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 terminal's type 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 (llN) must be included in the modes statement.

tab_string: "<string>";
The tab_string statement is optional. If present, it specifies a string, enclosed in quotes, to be sent to the terminal in raw mode in order to set all its horizontal tabs.

bauds: <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 in the bauds statement. 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
133
150
300
600
1200
1800
2400
4800
7200
9600

Delay Statements

Each delay statement is of the form:

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

The same number of values must be supplied as baud rates in the bauds statement. Each value specifies the number of delays to be used for the character described by the delay keyword (see below) 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 linefeed

horz_nl_delays
: the number of delays to be sent for each column position traversed by a carriage return

const_tab_delays
: the minimum number of delays to be sent with a horizontal tab

var_tab_delays
: the number of additional delays to be sent for each column position traversed by a horizontal tab

backspace_delays
: the number of delays to be sent with a backspace

vt_ff_delays
: the number of delays to be sent with a vertical tab or formfeed

Negative values for vert_nl_delays and backspace_delays have the same meanings as those described in the description of the set_delays order to the tty_ I/O module in the MPM Subsystem Writer's Guide. Values of 0 are assumed at all baud rates for any delay type not specified.

Example of bauds and delay statements

```
bauds:            110   150   300   1200   other;

vert_nl_delays:     2     3     6     24      30;
```

```
horz_nl_delays:      .2   .3   .5    2     5;

const_tab_delays:     0    1    2    7    10;

var_tab_delays:      .2   .3   .5    2     5;

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: <character>;

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

kill: <character>;

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

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 terminal's character code may be different from what was expected. At pres-

ent, only one possible preaccess message is defined, suitable for use with EBCDIC and Correspondence-code 2741 terminals; this mechanism may be generalized later. 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 terminal's answerback identification 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 provides compatibility with the present implementation, in which the answering service checks the ID for 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 or the word "none", 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 or the word "none", no output conversion table is used.

special: <table name>;
The special statement is optional. It specifies a table (defined by a special table entry) to be used as a special characters table when converting input and output (see the description of the special characters table entry, below). If it is omitted, or the table name is a null string or the word "none", 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 a table (defined by a translation table entry) used to translate input from the terminal's code to ASCII. If it is omitted, or the table name is a null string or the word "none", 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 terminal's code. If it is omitted, or the table name is a null string or the word "none", output is not translated.

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 ring 0; 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.

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 MCS that most closely corresponds to the terminal type described by this terminal type entry.

Global_Statements

A global statement specifies a default value for an attribute of a terminal type. It has the same form as the terminal type entry statement describing the attribute except that the statement keyword begins with a capital letter. A global statement may not appear within a terminal type entry. Global statements may be used for any statements included in a terminal type entry except for terminal_type, like, tab_string, tab_clear, tab_stop, and the delay statements. (A global Bauds statement is

allowed.)

Conversion_Table_Entry

A conversion table entry consists of two statements: one specifying the name of the table and one specifying its contents. A conversion table entry is described below.

conversion_table: <table name>;
<value0> <value1> ... <value127>;

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 tty_ I/O module in the MPM Subsytem Writer's Guide for a description of conversion tables and the indicators they contain.

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> ... <value127>;

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.).

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 0 to

three octal numbers of up to three digits each, separated by white space, representing a sequence of characters to be output to fulfill the specified function. All statements are required unless otherwise stated. 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. See the description of the various tables in the discussion of orders to the tty_ I/O module in the MPM Subsystem Writer's Guide 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 is optional. It specifies the sequence to be output to fulfill a "printer_on" order. The sequence is specified in the terminal's character code. If this statement is omitted, a null sequence is assumed, implying that the printer_on feature is not supported.

printer_off: <sequence>;
The printer_off statement is optional. It specifies the sequence to be output to fulfill a "printer_off" order. The sequence is specified in the terminal's character code. If this statement is omitted, a null sequence is assumed, implying that 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 is optional. It specifies the sequence to be output when output is suspended because the terminal's page length has been reached. If it is omitted, the character sequence "EOP" is assumed.

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> <sequence1>,
<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 is optional. It specifies those input characters that are to be interpreted as escape sequences when preceded by an escape character, and the result 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. This information is used by the answering service when a terminal dials up 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 baudN is a number representing a baud rate, or the word "any"; each line_typeN is the name of a valid line type, or the word "any"; each terminal_typeN 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.

Answerback_Table

The answerback table consists of entries specifying how to determine a terminal's 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 terminal's line type 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. 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) of the answerback string.

id N
The N characters starting at the scan pointer form the terminal's ID. 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 terminal's ID (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. If the type name is "none", the answerback is to be used to set the ID only, and the terminal type is not changed.

EXAMPLES

Sample_Terminal_Type_Entry

```
terminal_type:  TN300;
modes:  tabs,can,esc,erkl,^crecho,^lfecho,hndlquit,ll118;
tab_clear:  "\0332"; /* ESC 2 */
tab_stop:  "\0331          ";  /*  ESC 1 + 10 blanks */
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;
          /* No delays for CR or HT */
erase:  #;
kill:  @;
keyboard_addressing:  no;
old_type:  4;        /* old value for TermiNet */
input_conversion:  standard_input_conversion;
output_conversion:  ASCII_output_conversion;
special:  terminet_special;
input_translation: ;          /* this could have been omitted */
output_translation: none; /* this too */
```

Sample Use of Like Keyword

```
terminal_type:  TN80 like TN300;  /* same as 300 but ll = 80 */

modes:  tabs,can,esc,erkl,^crecho,^lfecho,hndlquit,ll80;
```

Sample Conversion Table

```
conversion_table:  ebcdic_output_conversion;

          07  07  07  07  07  07  07  07
          04  03  01  07  07  02  10  11
          07  07  07  07  07  07  07  07
          07  07  07  07  07  07  07  07
          00  00  00  00  00  00  00  0C
          00  00  00  00  00  00  00  00
          00  00  00  00  00  00  00  00
          00  00  00  00  00  00  00  0C
          00  00  00  00  00  00  00  00
          00  00  00  00  00  00  00  0C
          00  00  00  C0  00  00  00  0C
          00  00  00  21  00  22  00  00
          23  00  00  00  00  00  00  0C
          00  00  00  00  00  00  00  0C
          00  00  00  00  00  00  00  0C
          00  00  00  24  00  25  26  14;
```

Sample Special Characters Table

```
special_table:  ebcdic_special;

 new_line:  012;
 carriage_return: ;
 backspace:  010;
 tab:  011;
 vertical_tab: ;
 form_feed: ;

 printer_on: 015;   /* this is EBCDIC */
 printer_off: 016;  /* so is this */
```

```
 red_shift: 033 141; /* ESC a */
 black_shift: 033 142; /* ESC b */

 end_of_page: ; /* don't print EOP */

 output_escapes:
     21 134 074,     /* [ -> \< */
     22 134 076,     /* ] -> \> */
     23 134 047,     /* ` -> \' */
     24 134 050,     /* { -> \( */
     25 134 051,     /* } -> \) */
     26 134 164;     /* ~ -> \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,        /* \< -> [ */
     076 135,        /* \> -> ] */
     047 140,        /* \' -> ` */
     050 173,        /* \( -> { */
     051 175,        /*  \) -> } */
     164 176,        /* \t -> ~ */
     124 176;        /* \T -> ~ */
```

Sample Default Type Statement

```
default_types:        110  ASCII TTY33,
                      any  ASCII ASCII,
                      133   1050  1050,
                      133   2741  2741,
                     12C0    ARDS  ARDS,
                     1200 202ETX TN300,
                      any     any  G115;
```

Sample Answerback Table Entries

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

```
answerback:  match "0", id 3;
 type:  2741;

answerback:   search "XX", skip 3, match letter,
                    match digit, skip -2, id 4;
 type:  our_own;
```

APPENDIX

This appendix contains declarations of info structures associated with the following new orders: set_terminal_type, terminal_info, set_tabs, and set_terminal_data.

1. set_terminal_type

```
dcl 1 terminal_type_info,
      2 version fixed bin init(1),
      2 name char(16),
      2 flags aligned,
        3 set_tabs bit(1) unal,
        3 set_modes bit(1) unal,
        3 ignore_line_type bit(1) unal,
        3 mbz bit(33) unal;
```

2. terminal_info

```
dcl 1 terminal_info,
      2 version fixed bin init(1),
      2 term_type char(16),
      2 line_type fixed bin,
      2 baud_rate fixed bin,
      2 id char(4),
      2 pad(4) fixed bin;
```

3. set_tabs

```
dcl 1 tab_info,
      2 version fixed bin init(1),
      2 tab_string char(512) varying;
```

4. set_terminal_data

```
dcl 1 terminal_type_data,
      2 version fixed bin init(1),
      2 old_type fixed bin,
      2 name char(16),
      2 delay_ptr ptr,
      2 input_translation_ptr ptr,
      2 output_translation_ptr ptr,
      2 input_conversion_ptr ptr,
      2 output_conversion_ptr ptr,
      2 special_ptr ptr,
      2 editing_chars aligned,
        3 erase char(1) unal,
        3 kill char(1) unal,
        3 mbz fixed bin(17) unal,
      2 flags aligned,
        3 keyboard_addressing bit(1) unal,
        3 mbz bit(35) unal;
```