REMOTE TERMINAL CHARACTER STREAM PROCESSING IN MULTICS

by

J. H. Saltzer
Massachusetts Institute of Technology
Project MAC; and
Department of Electrical Engineering
Cambridge, Massachusetts

and

J. F, Ossanna
Bell Telephone Laboratories Incorporated
Murray Hill, New Jersey

March 10. 1970

PREPRINT

This is a preprint of a paper to be delivered at the 1970 Spring Joint Computer Conference, in Atlantic City, New Jersey, May 5-7, 1970.

REMOTE TERMINAL CHARACTER STREAM PROCESSING IN MULTICS

by

J. H. Saltzer and J. F. Ossanna

ABSTRACT

This paper describes system design and human engineering considerations pertinent to the processing of the character stream between a remote terminal and a general-purpose, interactive computer system.

The Multics system is used to provide examples of: terminal escape conventions which permit input of a full character set from a limited terminal, single character editing for minor typing mistakes, and reformatting of input text to produce a canonical stored form.

A formal description of the Multics canonical form for stored character strings appears in an appendix.

March 10, 1970

Introduction

There are a variety of considerations which are pertinent to the design of the interface between programs and typewriter-class remote terminal devices in a general-purpose time-sharing system. The conventions used for editing, converting, and reduction to canonical form of the stream of characters passing to and from remote terminals is the subject of this paper. The particular techniques used in the Multics* system are presented, as an example of a single unified design of the entire character stream processing interface. The sections which follow contain discussion of character set considerations, character stream processing objectives, character stream reduction to canonical form, line and print position deletion, and other interface problems. An appendix gives a formal description of the canonical form for stored character strings used in Multics.

Character set considerations

Although for many years computer specialists have been willing to accept whatever miscellaneous collection of characters and codes their syatems were delivered with, and to invent ingenious compromises when designing the syntax of programming languages, the impact of today's computer system is felt far beyond the specialist, and computer printout is (or should be) received by many who have neither time nor patience to decode information printed with inadequate graphic versatility. Report generation has, for some time, been a routine function. Recently, on-line documentation aids, such as RUNOFF [3], Datatext (IBM Corp.) or RAES (General Electric Co.) have attracted many users. Especially for

---

•Multics is a comprehensive genera1 purpose time-sharing system implemented on the General Electric 645 computer system. A general description of Multics can be found in reference [1] or [2].

the latter examples it is essential to have a character set encompassing both upper and lower case letters. Modern programming languages can certainly benefit from availability of a variety of special characters as syntactic delimiters, the ingenuity of PL/I in using a small set notwithstanding.

Probably the minimum character set acceptable today is one like the USASCII 128-character set [4] or IBM'S EBCDIC set with the provision that they be fully supported by upper/lower case printer and terminal hardware. The definition of support of a character set is almost as important as the fact of support. To be fully useful, one should be able to use the same full character set in composing program or data files, in literal character strings of a programming language, in arguments of calls to the supervisor and to library routines requiring symbolic names, as embedded character strings in program linkage information, and in input and output to typewriters, displays, printers, and cards. However, it may be necessary to place conversion packages in the path to and from some devices since it is rare to find that all the different hardware devices attached to a system use the same character set and character codes.

Character stream processing considerations

The treatment of character stream input and output may be degraded, from a human engineering point of view, unless it is tempered by the following two considerations:

1. If a computer system supports a variety of terminal devices (Multics, for example, supports both the IBM Model 2741 [5] and the Teletype Model 37 [6]) then it should be possible to work with any program from any terminal.

2. It should be possible to determine from the printed page, without ambiguity, both what went into the computer program and what the program tried to print out.

To be fully effective, these two considerations must apply to all input and output to the system itself (e.g., when logging in, choosing subsystems, etc.) as well as input and output from user programs, editors, etc.

As an example of the "device independence" convention, Multics uses the USASCII character set in all internal interfaces and provides standard techniques for dealing with devices which are non-USASCII. When using the GE-645 USASCII line printer or the Teletype Model 37, there is no difficulty in accepting any USASCII graphic for input or output from any user or system program. In order to use non-USASCII hardware devices, one USASCII graphic (the left slant) is set aside as a "software escape" character. When a non-USASCII device (say the IBM 2741 typewriter with an EBCDIC print element) is to be used, one first makes a correspondence, so far as possible, between graphics available on the device and graphics of USASCII, being sure that some character of the device corresponds to the software escape character. Thus, for the IBM 2741, there are 85 obviously corresponding graphics; the EBCDIC overbar, cent sign, and apostrophe can correspond to the USASCII circumflex, left slant, and acute accent respectively, leaving the IBM 2741 unable to represent six USASCII graphic characters. For each of the six missing characters a two character sequence beginning with the software escape character is defined, as shown in Table I. The escape character itself, as well

as any illegal character code value, is represented by a four character sequence, namely the escape character followed by a 3-digit octal representation of the character code. Thus, it is possible from an IBM 2741 to easily communicate all the characters in the full USASCII set.

A similar, though much more painful, set of escape conventions has been devised for use of the Model 33 and 35 Teletypes. The absence of upper and lower case distinction on these machines is the principal obstacle; two printed 2-character escape sequences are used to indicate that succeeding letters are to be interpreted in a specific case shift.

Note that consideration number two above, that the printed record be unambiguous, militates against character set extension conventions based on non-printing and otherwise unused control characters. Such conventions inevitably lead to difficulty in debugging, since the printed record cannot be used as a guide to the way in which the input was interpreted.

The objective of typewriter device independence also has some implications for control characters. The Multics strategy here is to choose a small subset of the possible control characters, give them precise meanings, and attempt to honor those meanings on every device, by interpretation if necessary. Thus, a "new page" character appears in the subset; on a Model 37 teletype it is interpreted by issuing a form feed and a carriage return; on an IBM 2741 it is interpreted by giving an appropriate number of new line characters.*

Of the 33 possible USASCI[control characters, 11 are defined in Multics as shown in Table II.

---

*This interpretation of the form feed function is consistent with the International Standards Organization option of interpreting the 1ine-feed" code as "new line" including carriage return.

ASCII Character Name	ASCII Graphic	Normal EBCDIC Escape	Alternate "edited" Escape.
Right Square Bracket	]	¢>	
Left Square Bracket	[	¢<	
Right Brace	}	¢)	
Left Brace	{	¢(	
Tilde	~	¢t	
Grave Accent	`	¢'	

TABLE I - Escape conventions for input and output of USASCII from an EBCDIC typewriter.

USASCII NAME	MULTICS NAME	MULTICS MEANING
BEL	BEL	Sound an audible alarm.
BS	BS	Backspace. Move carriage back one column. The backspace implies overstriking rather than erasure.
HT	HT	Horizontal Tabulate. Move carriage to next horizontal tab stop. Default tab stops are assumed to be at columns 11, 21, 31, 41, etc.
LF	NL	New Line. Move carriage to left edge of next line.
SO	RRS	Red Ribbon Shift.
SI	BRS	Black Ribbon Shift
VT	VT	Vertical Tabulate. Move carriage to next vertical tab stop. Default tab stops are assumed to be at lines 11, 21, 31, etc.
FF	NP	New Page. Move carriage to the left edge of the top of the next page.
DC2	HLF	Half-Line Forward Feed.
DC4	HLR	Half-Line Reverse Feed.
DEL	PAD	Padding Character. This character is discarded when encountered in an output line.

TABLE II USASCI Control Characters as Used in Multics

Red and black shift characters appear in the set because of their convenience in providing emphasis in comments, both by system and by user routines. The half-line forward and half-line reverse feed characters were included to facilitate experimentation with the Model 37 Teletype; these characters are not currently interpretable on other devices.

One interesting point is the choice of a "null" or "padding" character used to fill out strings after the last meaningful character. By convention, padding characters appearing in an output stream are to be discarded, either by hardware or software. The USASCII choice of code value zero for the null character has the interesting side effect that if an uninitialized string (or random storage area) is unintentionally added to the output stream, all of the zeros found there will be assumed nulls, and discarded, possibly leaving no effect at all on the output stream. Debugging a program in such a situation can be extraordinarily awkward, since there is no visible evidence that the code manipulating the offending string was ever encountered.

In Multics, this problem was considered serious enough that the USASCII character "delete" (all bits one) was chosen as the padding character code. The zero code is considered illegal, along with all other unassigned code values, and is printed in octal whenever encountered.

As an example of a control function not appearing in the character set, the printer-on/printer-off function (to allow typing of passwords) is controlled by a special call which must be inserted before the next call to read information. This choice is dictated by the need to get back a status report which indicates that for the currently attached device, the printer cannot be turned on and off. Such a status report can be returned as an error code on a special call; there would be no convenient

way to return such status if the function were controlled by a character in the output stream.*

Canonical Form for Stored Character Strings

Probably the most significant impact of the constraint that the printed record be unambiguous is the interaction of that constraint with the carriage motion control characters of the USASCII and EBCDIC sets. Although most characters imply "type a character in the current position and move to the next one," three commonly provided characters, namely backspace, horizontal tab, and carriage return (no line feed) do cause ambiguity.

For example, suppose that one chooses to implement an ALGOL language in which keywords are underlined. The keyword for may now be typed in at least a dozen different ways, all with the same printed result but all with different orders for the individual letters and backspaces. It is unreasonable to expect a tranalator to accept a dozen different, but equivalent, ways of typing every control word; it is equally unreasonable to require that the typist do his underlining in a standard way since if he slips, there is no way he can tell from his printed record (or later protestations of the compiler) what he has done wrong. A similar dilemma occurs in a manuscript editing system if the user types in underlined words, and later tries to edit them.

An answer to this dilemma is to process all character text entering the system to convert it into a canonical form. For example,

*The initial Multics implementation temporarily uses the character codes for USASCII ACK and NAK for this purpose, as an implementation expedient. In addition, a number of additional codes are accepted to permit experimentation with special features of the Model 37 Teletype; these codes may become standard if the features they trigger appear useful enough to simulate on all devices.

on a "read" call Multics would return the string:

_<BS>f_<BS>o_<BS>r

(where <BS> is the backspace character) as the canonical character string representation of the printed image of for independently of the way in which it had been typed. Canonical reduction is accomplished by scanning across a completed input line, associating a carriage position with each printed graphic encountered, then sorting the graphics into order by carriage or print position. When two or more graphics are found in the same print position, they are placed in order by numerical collating sequence with backspace characters between. Thus, if two different streams of characters produce the same printed image, after canonical reduction they will be represented by the same stored string. Any program can thus easily compare two canonical strings to discover if they produce the same printed image. No restriction is placed on the human being at his console; he is free to type a non-canonical character stream. This stream will automatically be converted to the canonical form before it reaches his program. (There is also an escape hatch for the user who wants his program to receive the raw input from his typewriter, unprocessed in any way.)

Similarly, a typewriter control module is free to rework a canonical stream for output into a different form if, for example, the different form happens to print more rapidly or reliably.

In order to accomplish canonical reduction, it is necessary that the typewriter control module be able to determine unambiguously what precise physical motion of the device corresponds to the character stream coming

from or going to it. In particular, it must know the location of physical tab settings. This requirement places a constraint on devices with movable tab stops; when the tab stops are moved, the system must be informed of the new settings.

The apparent complexity of the Multics canonical form, which is formally described in Appendix I, is a result of its generality in dealing with all possible combinations of typewriter carriage motions. Viewed in the perspective of present day language input to computer systems, one may observe that many of the alternatives are rarely, if ever, encountered. In fact for most input, the following three statements, describing a simplified canonical form, are completely adequate:

1. A message consists of strings of character positions separated by carriage motion.
2. Carriage motions consist of New Line or Space characters.
3. Character positions consist of a single graphic or an overstruck graphic. A character position representing overstrikes contains a graphic, a backspace character, a graphic, etc., with the graphics in ascending collating sequence.

Thus we may conclude that for the most part, the canonical stream will differ little with the raw input stream from which it was derived.

A strict application of the canonical form as given in Appendix I has a side effect which has affected its use in Multics. Correct application leads to replacement of all horizontal tab characters with space characters in appropriate numbers. If one is creating a file of

tabular information, it is possible that the ambiguity introduced by the horizontal tab character is in fact desirable; if a short entry at the left of a line is later expanded, words in that entry move over, but items in columns to the right of that entry should stay in their original carriage position; the horizontal tab facilitates expressing this concept. A similar comment applies to the form feed character.

The initial Multics implementation allows the horizontal tab character, if typed, to sneak through the canonical reduction process and appear in a stored string. A more elegant approach to this problem is to devise a set of conventions for a text editor which allows one to type in and edit tabular columns conveniently, even though the information is stored in strictly canonical form. Since the most common way of storing a symbolic program is in tabular columns, the need for simple conventions to handle this situation cannot be ignored.

It is interesting to note that most format statement interpreters, such as those commonly implemented for FORTRAN and PL/I, fail to maintain proper column alignment when handed character strings containing embedded backspaces, such as names containing over struck accents. For complete integration of such character strings into a system, one should expand the notion of character counts to include print position counts as well as storage position counts. For example, the value returned by a built-in string length function should be a print position count if the result is used in formatting output; it should be a storage location count if the result is used to allocate space in memory.

Line and Print Position Deletion Conventions

Experience has shown that even with sophisticated editor programs available, two minimal editing conventions are very useful for human input to a computer system. These two conventions give the typist these editing capabilities at the instant he is typing:

1. Ability to delete the last character or characters typed.
2. Ability to delete all of the current line typed up to this point.

(More complex editing capabilities must also be available, but they fall in the domain of editing programs which can work with lines previously typed as well as the current input stream.) By framing these two editing conventions in the language of the canonical form, it is possible to preserve the ability to interpret unambiguously a typed line image despite the fact that editing was required.

The first editing convention is to reserve one graphic, (in Multics, the "number" sign), as the erase character. When this character appears in a print position, it erases itself and the contents of the previous print position. If the erase follows simple carriage motion, the entire carriage motion is erased. Several successive erase characters will erase an equal number of preceding print positions or simple carriage motions. Since erase processing occurs after the transformation to canonical form, there is no ambiguity as to which print position is erased; the printed line image is always the guide. Whenever a print position is erased, the carriage motions (if any) on the two sides of the erased print position are combined into a single carriage motion.

The second editing convention reserves another graphic, (in Multics, the "commercial at" sign) as the kill character. When this character appears in a print position, the contents of that line up to and including the kill character are discarded. Again, since the kill processing occurs after the conversion to canonical form, there can be no ambiguity about which characters have been discarded. By convention, kill is done before erase, so that it is not possible to erase a kill character.

Other Interface Conventions

Two other conventions which can smooth the human interface on character stream input and output are worth noting. The first is that many devices contain special control features such as line feed without carriage movement, which can be used to speed up printing in special cases. If the system-supplied terminal control software automatically does whatever speedups it can identify, the user is not motivated to try to do them himself and risk dependence on the particular control feature of the terminal he happens to be working with. For example, the system can automatically insert tabs (followed by backspaces if necessary) in place of long strings of spaces, and it also can type centered short tabular information with line feed and backspace sequences between lines.

The second convention has been alluded to already. If character string input is highly processed for routine use, there must be available an escape by which a program can obtain the raw, unconverted, non-canonical, and unedited keystrokes of the typist, if it wants to. Only through such an escape can certain special situations (including experimenting with a

different set of proposed processing conventions) be handled. In Multics, there are three modes of character handling--normal, raw, and edited.* The raw mode means no processing whatsoever on input or output streams, while the normal mode provides character escapes, canonical reduction, and erase and kill editing. The edited mode (effective only on output if requested) is designed to produce high quality, clean copy; every effort is made to avoid using escape conventions. For example, illegal characters are discarded and graphics not available on the output device used are typed with the "overstrike" escapes of Table I, or else left as a blank space so that they may be drawn in by hand.

Conclusions

The preceding sections have discussed both the background considerations and the design of the Multics remote terminal character stream interface. Several years of experience in using this interface, first in a special editor on the 7094 Compatible Time-Sharing System and more recently as the standard system interface for Multics, have indicated that the design is implementable, usable and effective. Probably the most important aspect of the design is that the casual user, who has not yet encountered a problem for which canonical reduction, or character set escapes, or special character definitions are needed, does not need to concern himself with these ideas; yet as he expands his programming objectives to the point where he encounters one of these needs, he finds that a method has been latently available all along in the standard system interface.

---

*The "raw" mode is not yet implemented.

There should be no assumption that the particular set of conventions described here is the only useful set. At the very least, there are issues of taste and opinion which have influenced the design. More importantly, systems with only slightly different objectives may be able to utilize substantially different approaches to handling character streams,

Acknowledgments

Many of the techniques described here were developed over a several year time span by the builders of the 7094 Compatible Time-Sharing System (CTSS) at MIT Project MAC, and by the implementers of Multics, a cooperative research project of the General Electric Company, the Bell Telephone Laboratories, Inc., and the Massachusetts Institute of Technology.

The usefulness of a canonical form for stored character strings was independently brought to my attention by E. Van Horne and C. Strachey; they had each implemented simple canonical forms on CTSS and in the TITAN operating system for the ATLAS computer, respectively. F. J. Corbato' and R. Morris developed the pattern of escape sequence usage described here. Others contributing to the understanding of the issues involved in the character stream interface were R. C. Daley, S. D. Dunten, and M, D. Mcllroy.

Work reported here was supported in part by the Advanced Research Projects Agency, Department of Defense, under Office of Nava1 Research Contract Nonr-4102(01).

APPENDIX I

The Multics Canonical Form

To describe the Multics canonical form, we give a set of definitions of a canonical message. Each definition is followed by a discussion of its implications. PL/I-style formal definitions are included for the benefit of readers who find them useful [7]. Other readers may safely ignore them at a small cost in precision. In the formal definitions, capitalized abbreviations stand for the control characters in Table II.

1. The canonical form deals with messages. A message consists of a sequence of print positions, possibly separated by, beginning, or ending with carriage motion.

message ::= [carriage motion] $\left[\right.$[print position]...[carriage motion]$\left.\right]$ ...

Typewriter input is usually delimited by action characters, that is, some character which, upon receipt by the system, indicates that the typist is satisfied with the previous string of typing. Most commonly, the new line character, or some variant, is used for this function. Receipt of the action character initiates canonical reduction.

The most important property of the canonical form is that graphics are in the order that they appear on the printed page reading from left to right and top to bottom. Between the graphic characters

appear only the carriage motion characters which are necessary to move the carriage from one graphic to the next. Overstruck graphics are stored in a standard form including a back space character (see below).

2. There are two mutually exclusive types of carriage motion, gross motion and simple motion.

carriage motion ::= { gross motion | simple motion | gross motion simple motion }

Carriage motion generally appears between two graphics; the amount of motion represented depends only on the relative position of the two graphics on the page. Simple motion separates characters within a printed line; it includes positioning, for example, for superscripts and subscripts. Gross motion separates lines.

3. Gross motion consists of any number of successive New Line (NL) characters,

gross motion ::= { NL } ...

The system must translate vertical tabs and form feeds into new line characters on input.

4. Simple motion consists of any number of Space characters (SP) followed by some number (possibly zero) of vertical half-line forward (HLF) or reverse (HLR) characters. The number of vertical half line feed characters is exactly tlie number needed to move the carriage from the lowest character of the preceding print position to the highest character of the next print position.

simple motion ::= [SP] ... [[HLF]... | [HLR]...]

The basis for the amount of simple carriage motion represented is always the horizontal and vertical distance between successive graphics that appears on the actual device. In the translation to and from the canonical form, the system must of course take into account the actual (possibly variable) horizontal tab stops on the physical device.

In some systems, a "relative horizotal tab" character is defined. Some character code (for example, USASCII DC1) is reserved for this meaning, and by convention the immediately following character storage position contains a count which is interpreted as the size of the horizontal white space to be left. Such a character fits smoothly into the canonical form described here in place of the successive spaces implied by the definition above. It also minimizes the space requirement of a canonical string. It does require some language features, or subroutines, to extract the count as an integer, to determine its size. It also means that character comparison is harder to implement; equality of a character with one found in a string may mean either that the hoped for character has been found or it may mean that a relative tab count happens to have the same bit pattern as the desired character; reference to the previous character in the string is required to distinguish the two cases.

5. A print position consists of some non-zero number of character positions, occupying different half line vertical positions in

the same horizontal carriage position. All but the last character position of a print position are followed by a backspace character and some number of HLF characters.

print position ::= character position [BS [HLF] ... character position] ...

6. A character position consists of a sequence of graphic formers separated by backspace characters. The graphic formers are ordered according to the USASCII collating sequence of the graphics they contain. (The first graphic former contains the graphic with the smallest code, etc.) Two graphic formers containing the same graphic will never appear in the same character position.

character position ::= graphic former [BS graphic former] ...

Note that all possible uses of a backspace character in a raw input stream have been covered by statements about horizontal carriage movements and overstruck graphics.

7. A graphic former is a possibly zero-length setup sequence of graphic controls followed by one of the 94 USASCII non-blank graphic characters.

graphic former ::= [setup sequence] { one of the 94 USASCII graphic characters }

8. A graphic setup sequence is a color shift or a bell (BEL) or a color shift followed by a bell. The color shift only appears when the following graphic is to be a different color from the preceding one in the message.

setup sequence ::= { [RRS / BRS] [BEL] / BEL }

In the absence of a color shift, the first graphic in a message is printed in black shift. Other control characters are treated similarly to bell. They appear immediately before the next graphic typed, in the order typed.

By virtue of the above definitions, the control characters HT, VT, and CR will never appear in a canonical stream.

REFERENCES

[1] Corbató, F. J., et.al., "A New Remote-Accessed Man-Machine System" AFIPS Conference Proceedings, 27 (1965 FJCC), Spartan Books, Washington, D.C., 1965, pp. 185-247.

[2] The Multip1exed Information and Computing Service: Programmer's Manual, M.I.T. Project MAC, Cambridge, Mass., 1969 (to be published).

[3] Saltzer, J, H,, "Manuscript Typing and Editing," in The Compatible Time-Sharing System: A Programmer's Guide. 2nd Edition, M.I.T. Press, Cambridge, Massachusetts, 1965.

[4] "USA Standard Code for Information Interchange," X3. 4-1968, USA Standards Institute, October, 1968.

[5] "IBM 2741 Communications Terminal," IKM Systems Reference Library, Form A24-3415, IBM Corporation, New York.

[6] "Model 37 Teletypewriter Stations for DATA-PHONE Service," Bell System Data Communications Technical Reference, American Telephone and Telegraph Company, New York, September, 1968.

[7] "PL/I Language Specifications," IBM System Reference Library, Form C28-6571, IBM Corporation, New York.