Changes since last [illegible]

Duxbury Systems, Inc.
P.O. Box 1523
Duxbury, Massachusetts 02332
U.S.A.

August 11, 1976

DS-JES-760811

Summary of Changes to Duxbury Braille Translation System

Release 8/9/76

Translator

Fixes

1. Spurious fill characters should no longer appear at the beginning of occasional paragraphs.

2. Tab stop setting control symbols ($stb...) should now be recognized properly.

Enhancements

1. The control symbol $HDS has been extended to allow conditional page eject before the first line of the heading, unless at least n lines will be left on the current page after the first line of the heading. The number of lines required is attached directly to the $HDS symbol, e.g. $hds10. If no number is given, 1 is assumed. The operation of the present $HDS symbol is realizable by $hds0.

2. A new control symbol, $CTB, has been added. This symbol clears all tabs, i.e. all the stops are set to right-adjust at the far right of the page. This should make it easier to use the automatic tab feature for table layouts by providing a single means of removing unwanted current tab settings, including the initial settings.

3. A centering tab type has been added to the existing left-, right-, and decimal-point alignment types. The designator C may be used wherever L, R, or D may be used (i.e. in $STB, $TAB or #n controls), the meaning being that words so tabbed are aligned with centers falling on the specified column.

4. The Translator will now ask for an output file for remarks; the file name or channel number should be specified in the usual way. This file will receive a copy of the remark (exception message) output as on the main console ($TTO). If the channel number -1 is given, no such file will be written.

5. A new exception message of the form
   OP xxx NEAR IAL xxxx
   will now be generated each time the translator starts on a braille page divisible by 10. As with the other messages, OP means "output page", IAL "input absolute line." With 25-line, 40-character-wide pages, this message should appear about every 1 1/2 minutes, or every 200 input lines. Its purpose is to serve as reassurance at the console that the Translator is running, and an indicator of progress; in the auxiliary message file, the

message will serve as a braille vs. input cross-reference index.

6. A new error message, "CHAR. OUTSIDE RANGE", has been added in place of the (misdocumented) "EXCESS CHARACTER IGNORED" message. This latter message's meaning should be altered in the document; it actually implies that an excessively long word (group of contiguous nonblank characters) has been encountered in the input, and that characters beyond the 40th in such a sequence have been ignored.

7. The control symbol $PTYS (poetry start) has been extended to allow specification of the runover line indentation amount. The number of spaces to be left blank is specified by a number attached to the symbol, e.g. $PTYS4. If no such number is given, 2 is assumed, corresponding to the present $PTYS symbol. Note that, unlike $tls ... $tle and $hds ... $hde, $PTYS and $PTYE symbols need not necessarily occur in pairs. In fact, $PTYE is equivalent to $PTY0. Thus successive $PTYSn symbols may be used to vary the runover line indentation amount. This facility, together with the $INDn symbol (whereby all lines, i.e. the left margin, may be indented) permits different combinations of indentation for many purposes besides poetry. Program logic schematics, for example, with subordinate clauses indented varying amounts to indicate depth of nesting, come to mind.

8. The number of braille lines per page has been arbitrarily limited to 100, to guard against accidentally setting up so that an unreasonable number of empty lines could be generated in response to a $PG (new page) command.

Editor Enhancements

1. A new program, DEPRIN, has been added for quick printing of files produced by [illegible] to be used by the Duxbury Editor. It is initiated by the CLI command DEPRIN. It asks for a free channel to use for reading the file(s) to be printed. (It seems unfortunate to raise the concept of a channel to the user level, but this seems safest in the presence of uncertainty over how separated the foreground is from the background.) It then asks for the file name, to be entered in the usual way. Any open errors are reported; otherwise the file is printed just as by DEDIT except that the word PRINTED appears to the left of the date in the heading. The program then asks for another file, and so forth until the operator responds ".." to the name request, whereupon the program quits.

2. The file mode is now printed at the head of the line-numbered listing by both DEDIT and DEPRIN, and moreover the mode is passed as the first argument in calls to DEPL, the print-line subroutine. DEPL1, the version of DEPL used for upper-case-only printers not capable of

overstriking, has been modified so as not to print upper case indicators below the lines of a mode 1 (u.c. only) file.

EPROOF Enhancement

1. EPROOF will now ask for the number of lines per braille page. (Previously, 25 was assumed.)

Table Changes

1. An L class character (letter or equivalent) is now required as right context to the space after "to" and "into" as well as "by" before these words will be contracted and brought up against the words following them.

2. A period followed by a space is now always interpreted as a period in braille, even when following numbers.

3. The word "insofar" should now translate correctly.

4. No space should follow the opening quote that is automatically added at the beginning of paragraphs within quoted material.

Observed Anomalies

1. B, C, W and w do not work as isolated-letter commands when preceded by blank
   — Append letter sign manually (+)

   } Use fixes

2. Just once (in many trials) the editor has been observed to truncate a line during execution of the REPLACE command. Error repeated on the identical situation, but not when LI was used
   — Use LI with RE Cmnd, or check output.

   } Use fixes

3. Because of puzzling behaviour of the FORTRAN S-format read, backspace (R) cannot be entered on the editor.
   — $D has been entered in the tables as a temporary equivalent to \. Or, use the system editor.

Known Limitations

1. EPROOF expects 25-line ~~pages~~ bailla pages only. [Parameter easily added]
2. Unit-of-measure to be input without terminal period — e.g. lbs. → lbs

✓ = done

Contemplated Improvements/Changes

✓1. Allow $HDS n — goes to new page if ∄ at least n lines (after current) on the page. $HDS ≡ $HDS1
current $HDS ≡ $HDS0
Similarly, $CPn (conditional page if not ≥ lines left)

2. $PPn (add n to current pg. no.)

Doc in { Updates / Errors } - DEDIT

Docin { Update / Errors } – TRANSL

pg.

✓ 3-20 Expl. of "EXCESS CHARACTER IGNORED" – Should be:
"An input word (sequence of non-blanks between blanks) exceeds 40 characters in length. Characters beyond the 40th are ignored."

~~3-21 Parenthetical ref. to #3 should be deleted~~

Doc'n { Update / Errors - EPROOF

Doc'n { Update / Errors } – PREPTB

~~12~~

✓ REFS – Should be as Sullivan (75) in TRANSL doc,
and
J.E. Sullivan, The Duxbury Braille Translator
User's Manual, Duxbury Systems, Inc. document
#DS-UES-760716 (July 1976)

THE DUXBURY EDITOR USER'S MANUAL

DS-D8561-0

Ordering No. DS-D8561
Supersedes: DS-JES-760704

123 Lowell Drive
Stow, Massachusetts 01775 USA

NOTICE

Original Release May, 1978

Written for Duxbury Systems, Inc. by Joseph E. Sullivan

TABLE OF CONTENTS

1. INTRODUCTION

SCOPE OF APPLICATION

The Duxbury Editor is a general-purpose program for creating and altering files containing source code, natural language text, or any similar source data. It is "line-oriented"; that is, the basic unit of reference for operations on the file is a group of characters terminated by a carriage return. Text may be upper-case only (e.g., most source program files) or mixed case (e.g., English text). The console device used to run the Editor may be upper-case-only (e.g., a typical teletype) or a mixed case terminal. Either type of file may be edited from either type of terminal. A previously-prepared file may also be used to supply commands to the Editor instead of a console terminal, if desired.

The Editor is designed as a reentrant subroutine, which means that it can readily be used in multitasking situations-- for example where editing is to take place simultaneously at several terminals.

OPERATIONAL CONCEPT

The Editor operates directly upon a direct-access file, that is, a disk-resident file in which individual lines may be accessed easily in any order. There is one such direct-access file for each body of text to be edited. These direct-access files are in a special format not generally used by programs other than the Editor itself, however. For this reason, the Editor provides a facility for converting easily from direct access to sequential form and vice versa. Sequential files permit lines to be accessed conveniently only in order; however, they are the most common type of file used and may reside on many media (e.g. magnetic tape, paper tape) besides disk.

Thus the user typically creates a file initially in direct access form, and converts it to a separate sequential file for use. If changes are then found to be required, the original direct access file may then be edited and reconverted to a sequential file, superseding the old. This cycle may be repeated as often as necessary, although typically it will be found desirable to go the other way and reconstitute the direct access file from the sequential file (principally to "clean up" the line numbers, as will be seen) -- at least on occasion. The approach of having two essentially equivalent files extant has the advantage that they act as mutual backup, although of course disk space suffers as a result. Usually, once a file has become stable, the direct access file is deleted and set up again only temporarily for editing as necessary.

When a new file is to be created, the user simply enters lines after giving the "accept" command. (The commands are described in detail in a later section.) When an old file is to be altered, the user is presumed to be working from a line-numbered listing of the current file with edit markings, or at least a marked-up plain listing supplemented by a line-numbered listing for ready comparison. It is possible, by using the "replace" command for searching, to locate text without knowledge of the corresponding line number; this is not, however, an efficient technique for extensive editing, either from the computer's standpoint or from the user's. A line-numbered listing for a direct-access file can be obtained by directing a "put" command to the line printer. The utility DEPRIN (described in appendix A) can be used to obtain a line-numbered listing of a sequential file. Note that line numbers are for reference purposes only and are not part of the text data itself (although it is possible to make the direct-access file numbers part of the sequential file data, if desired).

The user refers to the lines he wishes to alter, delete, insert or move about by line number. For example, the command

```
15 to 20 replace Tom by Harry
```

would alter lines 15 through 20 in such a way that wherever the three-character sequence "Tom" used to appear "Harry" will appear instead. As long as a particular direct-access file is used, the original line numbers are preserved in their relationship to the data itself, even though lines are deleted or groups of lines are moved about, so that it remains easy to refer to the text working from the marked-up original listing. Any new (inserted) lines are serially assigned new numbers, regardless of the position in which they are inserted. The net effect of these policies is that line numbers, which are originally in order, may become quite out of sequence as the editing progresses. The Editor keeps track of the true sequence of lines by a "map," e.g.:

```
12-93
199-213
1-11
214-234
```

which would define the line sequence to be:

```
12, 13, 14, ... 92, 93, 199, 200, ... 212, 213, 1, 2,
3, ... 10, 11, 214, 215, ... 233, 234
```

This map is available to the user at all times to aid in locating lines, although if the user is working from a line-numbered listing the line numbers are simply names whose numerical value is of little consequence. Conversion to sequential file form and back again has the effect of renumbering the lines. This is necessary when the map becomes too long (i.e. has too many dis-

continuities) for the Editor's capacity, and may well be desirable before that point, depending on the use being made of the line numbers.

METHOD

The Editor assigns each line to a particular 128-character (1/4 disk block) area in the direct-access file, the position of which area is a simple function of the line number. The line sequence is defined by the map, as explained above, which is kept in the first few file records. Thus the lines themselves are never moved, and many operations (such as deletions) are accomplished simply by manipulating the map.

In addition to current lines in the file, the last set of lines removed by a "disjoin" command is "remembered" in a separate part of the map, against the possibility that the user will "join" them to the set of current lines. Including these remembered lines, there may be at most 62 continuous blocks of lines in the map at one time. (The Editor automatically erases spurious discontinuities, as for example when a block beginning with line 12 is joined after a block ending with line 11.) The dead space occupied by lines that have been disjoined and not rejoined is not reused for other lines. This has the disadvantage of wasting file space but the advantage that a command, i.e. "summon," can bring back to life lines that have been deleted in error.

2 USAGE

PRINCIPLES OF OPERATION

Input-Output Interface

The Editor is a FORTRAN subroutine to be called with certain parameters to determine its mode of operation, as explained more fully under "Calling the Editor." Among other things, these parameters determine four I-O channels that will be used by the Editor as follows:

(1) terminal output, i.e. prompting for console terminal input data and commands, and responses to commands (e.g. error messages);
(2) terminal input, e.g. commands;
(3) the direct-access file; and
(4) sequential file(s).

It is possible to assign channel (2) to a disk or other file containing prestored commands to drive the Editor. Normally, however, (1) and (2) refer to the same "live" terminal; the rest of this description will assume that this is the case, as the Editor was primarily designed for such interactive use.

With allowance for the presence of line numbers and possible breaking of the line because of terminal width, the relationship between the line as it appears in the file and as it appears on the terminal is direct, i.e., a simple copy, if the modes (upper case or mixed) are the same. If the file is upper-case-only and the terminal is mixed, then output (from the file to the terminal) will be upper case but either case may be used for input: lower case letters will be converted to upper case. In the reverse mismatch situation, lower-case letters in the file are represented as simple upper-case letters on the terminal and upper-case letters and slashes (/) in the file are represented on the terminal as a slash followed by the character, on both input and output. For example:

Terminal	File
AB	ab
A/B	aB
A//B	a/b
A///B	a/B
A////B	a//b
A/////B	a//B

(On input, a slash preceding a character other than a letter or slash is simply ignored.)

Edit Cycle

The Editor performs the following operations in sequence:

(1) The Editor announces itself with the caption

```
DUXBURY EDITOR 7/77
ARGS ...
```

where ... represents the arguments passed to the Editor by call (see "Calling the Editor"). These values are usually not of interest to the terminal operator, unless he wishes to check that the right characteristics have been assigned to his terminal.

(2) If any of the argument values are out of valid range, the editor will ask for them to be supplied from the terminal, as explained under "Calling the Editor." Again, this would not normally be required of the terminal operator.

(3) The Editor asks for the direct-access file name:

```
FILE?:
```

This illustrates the general practice of putting out "?:" whenever a user response is required. A response to a question of this sort is always a sequence of characters without any embedded blanks (except for commands), followed by a carriage return.

The response in this case should be the name of the direct access file to be edited; it need not pre-exist. The format of the name is as prescribed by the operating system. If the file does not pre-exist, the Editor will issue the comment

```
* NEW FILE
```

When all files have been edited, the special response

```
..
```

to the "FILE" question will cause the Editor to return to the calling program.

(4) The Editor then asks:

```
FILE MODE?:
```

The response should be 1 for upper-case-only files, 2 for mixed case files.

(5) The Editor then accepts commands, preceding each by simply

?:

The "end" command will break this cycle, cause the file to be closed, and return to the "FILE" question (3) for another file or to close out operation of the Editor.

COMMANDS

Line Numbers

As was previously noted, line numbers are artificial tags attached to the data lines for identification purposes only. Although sequentially assigned originally, these numbers may no longer be in sequence in relation to the logical sequence of the lines after a series of moves and insertions. When addressing a line or lines for command purposes, either of the forms

x to y

or

x

may be used, where x and y represent numbers and the latter form is always equivalent to

x to x

In some commands, it may be meaningful to reference a dummy line preceding all lines in logical sequence. This dummy line always the line number 0.

Unless noted otherwise in the command description, line numbers x and y must both be current lines (or 0 where meaningful) and y must logically follow x in the file (though it may be numerically less than x).

The special words FIRST and LAST may be used, respectively, in place of the logically first and last line currently in the file.

Command Words

In the description that follows, the term "quote" refers to the double quote sign (") and the term "doubled quote" refers to two of these in succession ("").

All "words" in commands have one of two forms:

(1) a character other than a blank or quote followed by zero or more non-blank characters (i.e., up to the next blank);
(2) a quote, followed by zero or more characters up to the next quote that is not doubled. Quotes within the word are doubled.

In the first case, the "word" is the entire string of characters; in the second, the word is the string with the initial and final quotes deleted and any interior quotes undoubled.

Within a command, any word -- command name, data word or line number -- may be expressed in either form unless it is null (consists of no characters) or has embedded blanks or quotes. In the latter case, the second, quoted, form must be used. As a practical matter, only data words (as in the "replace" command) would need to be quoted, e.g.:

```
15 to 20 replace "to and fro" by ""
```

would simply delete the characters "to and fro" wherever they appeared in lines 15 through 20.

Command Formats

Commands are given by typing one or more command words, separated by one or more blanks, on a single line. Words beyond those needed to specify a command are generally ignored, although no more than than eight words are permitted.

In the descriptions that follow in the next section, brackets are used to surround words or word sequences that may or may not appear; e.g. line numbers, already described, are usually in the form

```
n1 [TO n2]
```

A comma within brackets denotes that either or neither (but not both) options may be selected, e.g.,

```
[LISTED, UNLISTED]
```

Both keywords (such as LISTED and TO, above) and command names may be abbreviated to their first two letters only, and either upper or lower case may be used in entering them although upper case is used th the descriptions. Lower-case n's and d's, either of which may be subscripted for distinction, are used in the descriptions to stand for line numbers and data words respectively. The case of data words is important if the file is mixed case (mode 2).

Command Descriptions

Accept

The command

n ACCEPT

with n equal to 0 or an existing line number, causes the acceptance (insertion) of new lines after line n. The Editor responds

.. ENDS INPUT:

Lines are then accepted from the terminal until the line consisting of only two periods and no other characters (not even blanks) is entered. Such a line is not considered data. If any data was accepted, i.e. if there were any lines before the ".." line, the Editor responds

* NEW LINES x - y

where x and y are the first and last (sequential) line numbers assigned the new lines. Logically, these lines follow line n. If n = 0, the new lines will precede all prior data.

In the unlikely event that a line comprising exactly two periods should need to be entered into a file, this can be accomplished by entering some other character or word and then using "replace" to change it to the desired two periods.

Disjoin

The command

n1 [TO n2] DISJOIN

causes line n1 or lines n1-n2 to be removed from the map of current lines but "remembered" in a temporary memory for possible subsequent JOIN. This temporary memory is overwritten by any subsequent DISJOIN or SUMMON command, and emptied (set to "no lines") by any JOIN command. Thus DISJOIN may be used for deleting lines, or in conjunction with JOIN for moving text.

End

The command

END

closes the current direct-access file; the Editor returns to the "FILE?:" query for possible additional files to edit. The closed file may later be reopened in the same state -- i.e. same line numbers, same map, and same "remembered" lines -- as pertained at the time of closing.

Get

The command

n GET filename

causes the sequential file whose name is given to be opened and read in so as to follow line n in logical sequence. The Editor responds

* NEW LINES x - y

as with the ACCEPT command, to inform the user what line numbers have been assigned. As an example, "0 GET XXX.S" would append the contents of file XXX.S at the beginning of the current file being edited. The GET command may be used for standard text ("boilerplate") insertion and for conversion of files from sequential to direct-access form for editing.

Join

The command

n JOIN

logically inserts the group of lines removed by a previous DISJOIN or recalled by a previous SUMMON (i.e. the present contents of the temporary memory) after line n. The memory is then cleared. JOIN thus is normally the second half of a DISJOIN-JOIN or SUMMON-JOIN combination. It cannot be used to create multiple copies of text; to do that, one would use a PUT followed by several GETs.

Map

The command

MAP [PAUSING, SUMMARY]

causes the current map to be listed on the terminal. Included are the map version number, highest line number currently in use, highest line as of the last map update in the file (which should be the same as the previous number), and counts of the active, remembered, and free (unused) line number blocks (i.e. map slots, of which there may be no more than 62). Total line counts for active and remembered lines are also given.

Normally a detailed listing of the line sequence (see "Operational Concept") is also given; this is omitted if the SUMMARY option is selected. If PAUSING is selected, the EDITOR pauses after each line of the detail listing to await one of the following instructions from the terminal:

D (or nil, i.e. just carriage return)
E

D or nil causes the Editor to proceed to the next line of listing; E causes it to terminate execution of the command.

Put

The command

n1 [TO n2] PUT filename [NUMBERED] [APPEND, DELETE]

causes lines n1 through n2 to be written out in the sequential file named. (They also remain in the direct access file being edited.) This command can be used for copying or saving portions of files, creating the equivalent sequential file for the entire file, or for generating listings (e.g. by naming $LPT as the destination file).

The NUMBERED option causes each line to be prefixed by its line number. If the APPEND option is used, the lines are written to the end of the existing file, after other data already present. If the DELETE option is used, the existing sequential file is deleted and created anew before being written. The APPEND and DELETE options are, of course, mutually exclusive. Neither should be used when writing to devices such as $LPT.

Replace

The command

n1 [TO n2] REPLACE d1 BY d2 [UNLISTED, LISTED, VERIFY]

causes all instances of the word d1 to be changed to d1 in lines n1 through n2. Either "word" may be a general string in either of the forms described under "Command Words."

Normally, i.e. if no options are used, the Editor lists just the line numbers where replacement occurred, together with the number of replacements for each line (unless the count of replacements is 1, in which case it is omitted). If the UNLISTED option is used, even this summary is omitted; replacement proceeds without feedback as to where replacement is occurring. If LISTED is used, the line number, count and line itself (after replacement) is listed for each line in which replacement occurred. Finally, if VERIFY is used, the EDITOR behaves as with LISTED but stops after displaying each changed line to await one of the following instructions:

Y (or nil, i.e. just carriage return) - yes, change and proceed
N - no, do not change (leave line as it was) and proceed
E - do not change, and exit from command execution
F - change, and exit from (finish) command

Note that replacement is made without rescanning any part of the string d2, so that in some cases instances of d1 may remain after replacement. For example:

... replace ata by ama

on the data line

atatata

would produce the data line

amatama

The string d2, but not d1, may be nil ("").

This command is useful for searching as well as for systematic substitution. One technique for doing this would be to replace with d2 = d1. Another would be to use the VERIFY option, answering N to each instruction request until the desired instance is found, thereupon answering F.

Summon

The command

n1 [TO n2] SUMMON

has the special purpose of recapturing lines that have been deleted -- i.e. disjoined and then subsequently lost from the temporary memory. For this command, n1 must be numerically less than or equal to n2, and n1 and n2 and all lines between must not be among the currently active lines of the file. The command causes lines n1 through n2 to be placed in the memory (thereby "forgetting" any current contents of the memory), for possible subsequent JOIN. Thus this command can be used to recover lines deleted in error, but it should be noted that any group of lines originally containing discontinuities will have to be reconstructed by not just one but a series of SUMMON-JOIN command pairs.

Type

The command

n1 [TO n2] TYPE [PAUSING] [NUMBERED]

causes the Editor to list the lines from n1 through n2 on the user's terminal. Normally, line numbers are not listed but if the NUMBERED option is used line numbers are listed above the corresponding line. If PAUSING is used, the Editor stops after each line displayed for further instructions as follows:

D (or nil, i.e. just carriage return) - down 1 (i.e. proceed)
U - up 1 (i.e. go back to the previous line)
E - exit from command execution

Remarks and Error Messages

The Editor comments on an unusual, and perhaps erroneous, response or command thus:

REMARK a P'R b

where a is the remark number, i.e. a key to the table below, and b is a parameter that may give additional information, according to the remark. The parameter's meaning, where there is one, is given in brackets in the table. When the parameter has no meaning, it is generally displayed as 0.

Except as noted, the Editor terminates command processing immediately upon discovery of the condition remarked upon.

Remark No.	Interpretation [parameter meaning, if any]
1	File opening error [error no. as defined by system OPEN]
2	Logic error within DEDIT [internal section no.]
3	Too many command words [maximum number allowed]
4	Command word too long (usually due to unbalanced quotation marks) [maximum word length]
5	"From" line (n or n1) not 0 or among current active lines [from-line number]
6	"To" line (n2) not greater than 0. [to-line number]
7	Unrecognized command name [decimal code for first two characters of command given]
8	"To" line (n2) not after n1 in file. [to-line number]
9	BY missing in REPLACE command
10	Nil initial word (d1) in REPLACE command
11	A line would overflow, i.e., exceed maximum allowable length, if replacement were to proceed. REPLACE execution is terminated. [line number of line in question]
12	"From" line number not completely a number or FIRST or LAST (with at least one line in the file) [line number as partially interpreted]
13	"To" line number not completely a number or FIRST or LAST (with at least one line in the file) [line number as partially interpreted]
14	Destination file opening error [error no. as defined by system OPEN]
15	Unused
16	Direct-access file write failure [error code]. Editor proceeds.
17	Unused

18 Error or end-of-file in terminal read [1=error, 2=EOF]. Line has not been received properly and should be reentered.
19 Line too long [maximum permissable length]. The line is truncated at the maximum length.
20 Direct access file reading error [error code as defined by system]
21 Version mismatch upon initial reading of map [version in map record 1]. A possible system "crash" during map write in previous session has truncated the map. Use SUMMON to recover any "lost" lines.
22 Inconsistent map -- probably because the file accessed is not an Editor format direct access file. The file is closed (unharmed) and a new file requested.
23 Unable to continue editing this file. (This remark always follows one or more others.) File closed and a new one requested.
24 Error on direct access file closure [system error code]. Editor proceeds.
25 Zero "from" line improper in this command
26 Improper number of words in command [no. of words]
27 Sequential file open error [system error code]
28 Sequential file reading error [line number]. Reading proceeds.
29 Sequential file closing error [system error code]. Editor proceeds.
30 Insufficient free group elements to initiate this command [number free]. Indicates too many line number discontinuities; sequential equivalent file should be generated and the direct access file reconstituted from it, thereby renumbering lines.
31 Direct-access file update-directory error [error code].
32 JOIN executed with no lines in the temporary memory.
33 Invalid option [word no.]
34 Line as typed or put truncated due to prefixing of line no. [original length]
35 Inappropriate entry during pause or verify [code of character entered]
36 In SUMMON command, n1 > n2 or n2 > highest line ever used [highest line ever used].
37 In SUMMON, line group overlaps currently active lines [1st line of currently active group overlapped.]
38 Line as displayed truncated [actual length]

3 TECHNICAL INFORMATION

CALLING THE EDITOR

The Editor is a reentrant FORTRAN subroutine. This is to allow it to be fit into custom online environments with many other kinds of tasks. It is called in the ordinary way, with five arguments, viz.:

```
CALL DEDIT (AACHNL, AALMXL, AACMOD, AAOPTL, AALINU)
```

The arguments are as follows:

AACHNL — Integer array, size 4. Elements are all channel numbers, in order: (1) console terminal write; (2) terminal read; (3) direct-access file(s); (4) sequential file(s). The two terminal channels should be open at the time of call and are left open on return; the other two should be closed at time of call and are left closed on return.

AALMXL — Integer expression, terminal maximum line size. Must be between 20 and 80 inclusive. May also be set to -1, in which case the value of line size and all following arguments are obtained from the terminal user directly, as explained in the next section.

AACMOD — Integer expression, console terminal case mode. 1 = upper case only, 2 = mixed case; other values illegal.

AAOPTL — Logical array, size 4. Elements are yes-no option selections, in order:

(1) The map version number is displayed each time the map is written to the file, even during the periodic updates under the ACCEPT command (see AALINU below). Otherwise (i.e., if the option is .FALSE.) the map version is not displayed most of the time. In general, this option should not be selected on slow terminals.

(2) All control characters detected on input from the terminal (other than those interpreted by system) are changed to spaces. Otherwise, control characters are not treated specially.

(3) Input lines from the console terminal may be "continued" so that more than one physical input line corresponds to one logical input line. With this option in effect, a line is continued by typing hyphen at the end, just before the carriage return. The hyphen and carriage return are in effect ignored for purposes of assembling a logical input line, whether it be a command or data. Several continuations may be used, if

necessary, to define a logical input line as long as the maximum of 80 characters per logical line is not exceeded. Otherwise, if this option is .FALSE., physical and logical input lines correspond one-for-one.

(4) Trailing spaces in input lines are ignored. Otherwise, they are considered part of the line.

AALINU Integer expression, lines between updates during ACCEPT. Must be >= 0. This controls how often the map and directory of the direct access file are updated to reflect all input to date while entering data under an ACCEPT command, that is, how many lines could at most be lost should system failure occur during data entry. If AAOPTL(1) is .TRUE. (see above), the map version is displayed at the point of each such update.

If any of the argument values are out of valid range, all the values (except for the channel nos.) are requested from the terminal user as described below.

PARAMETER SETTING FROM THE TERMINAL

If AALMXL is -1, or any of the arguments passed are out of valid value range, the Editor will, after announcing itself, ask the cryptic question:

WWWMVBCTIII?:

This is a request for the user to enter the parameters, other than AACHNL, in the same order as the argument list and in the exact format (I3, I1, 4L1, I3). The mnemonic-to-variable correspondence in the query is: W=width=AALMXL, M=mode=AACMOD, V=version nos.=AAOPTL(1); B=blanks=AAOPTL(2); C=continuation =AAOPTL(3); T=truncate trailing spaces=AAOPTL(4); I=interval between updates=AALINU.

The question is repeated until valid values are obtained from the terminal.

This facility is intended, obviously, for special use by programmers and not for general use. It permits, for example, a general "utility" editor to be generated that can be called up at any terminal and tailored on the spot to the characteristics of that terminal and the user's desires of the moment without any special programming.

"CRASH" RECOVERY

By the way in which the direct access file is structured and managed, there is almost no chance that a system hardware or

software failure during editing will leave the file in an inconsistent state. This means that recovery after such failure is normally simply a matter of reopening the file for editing. Lines entered under ACCEPT since the last map and directory update will, in general, be lost and will have to be reentered. If map versions were being displayed each file update, that data can be used to locate the point where reentry must commence; otherwise, it is generally easy enough to find that point by browsing with the TYPE command.

It is remotely possible that system failure could occur during the map writing process itself. Even if that should happen, the only effect should be that the map would be truncated, i.e. some groups of lines would appear to be lost even though they are, in fact, in the file and retrievable via the SUMMON command. Such an inconsistent condition should be automatically noticed by the Editor (see the description of remark #21).

Of course, certain kinds of system hardware and software failure can result in loss of any file and even whole disk packs; frequent backup of all system media to tape or removable disks, or some equivalent procedure, should be employed to permit reasonable recovery from such an eventuality.

EXPANDABILITY

Most of the numerical limits mentioned in the description, e.g. 62 map blocks, 80 characters line width, etc., are readily expandable -- though probably at some cost in memory of file space -- by altering compile-time parameters in the program and recompiling.

SUBROUTINES CALLED

DEDIT calls the following subroutines directly:

CAIF CLUC CQIF DERK IBYTE IULST LASTR MOVEB SBYTE

At this writing, the following is indirectly called:

LOCA (called by MOVEB)

APPENDIX A DEPRIN UTILITY

A separate utility program, activated by the CLI command DEPRIN, can be used to obtain line-numbered listings of sequential files. Such listings are particularly useful as edit masters, i.e. copies to be marked with all changes to be made in the next version.

DEPRIN first asks for a free channel number to use in reading the file(s). Respond "1" unless there is a special reason to choose another channel. DEPRIN then asks for the name of the sequential file to be listed. Any opening errors are reported; if there are none, the file is printed with a time-stamped header and line numbers to facilitate reference for editing. DEPRIN then asks for another file to be printed, continuing the cycle until ".." is given in response to the file name request, whereupon the program quits.

THE DUXBURY BRAILLE TRANSLATOR USER'S MANUAL

DS-JES-760716-00 01

Ordering No. DS-JES-760716

P. O. Box 1523
Duxbury, Massachusetts 02332

THE DUXBURY BRAILLE TRANSLATOR USER'S MANUAL

DS-JES-760716-01

Ordering No. DS-JES-760716

123 Lowell Drive
Stow, Massachusetts 01775
United States of America

NOTICE

Original Release July, 1976
Revised: October, 1976

Written for Duxbury Systems, Inc. by Joseph E. Sullivan

ABSTRACT

This document describes the Duxbury Braille Translator, a table-driven Fortran IV program for the translation of English text into Standard English (American) Braille, also known as grade 2 braille. Text in any of several foreign languages and English text to be transliterated as grade 1 braille or "computer" braille may also be handled. While general acquaintance with computer programming and the subject of braille translation would be helpful in reading the document, no special knowledge in these areas is presupposed. The program's method of operation, together with detailed instructions on using the program, are presented.

PREFACE

This document is intended to serve the needs of several different levels of interest. Those interested only in what the program will do, for example, need read only the introduction. Persons with a general interest in the subject of braille translation should add the section on method. Those who actually wish to use the program -- transcribers, editors, and typists -- will, of course, want to refer to the appropriate parts of the Usage Section and may or may not be interested in method.

This document corresponds to the 7/76 version (8/9/76 release) of the program, and the DUXSYS 7/76 (8/9/76 release) tables.

TABLE OF CONTENTS

1 INTRODUCTION

Braille, as it is used today in almost all contexts other than primer textbooks, is almost a system of shorthand and not simply a scheme for representing individual inkprint symbols in a tactile code. This system, called Standard English Braille (American) or grade 2 braille, is defined in EBAE [72]. A letter-for-letter transcription is called grade 1 braille.

The chief device used to reduce the number of braille signs in grade 2 is the contraction. A contraction is the representation of an inkprint letter-group or whole word by a relatively short sequence of braille signs; for example, the word "receiving" is represented by the four braille signs for r, c, v, and g in succession. There are 189 letter-groups that may be contracted.

Unfortunately (at least from the standpoint of automating the translation process), a given contractable letter-group is not necessarily contracted wherever it appears. Moreover, the rules governing the use of contractions frequently involve such matters as pronunciation and meaning. For example, in the word "disease," the "dis" and the "ea" are normally contracted. However, when this word is used in the (now obsolete) sense "lack of ease," the "ea" should not be contracted. This example, although admittedly farfetched, illustrates that in some circumstances even a human transcriber might have difficulty applying the rules. Cases routinely arise that are easy for a human transcriber, but still difficult for a mechanical process -- for example, distinguishing the musical note "do" from the verb "do." For these reasons, automatic natural-language to braille translation algorithms tend to be heuristic, which is to say fallible.

The Duxbury Braille Translator (hereinafter usually called the Translator) is a computer program embodying such a natural-language-to-braille translation algorithm. It is an outgrowth of two earlier programs, DOTSYS II and III, described in Millen [70A] and [70B], and Sullivan [75]. The basic translation algorithm remains similar, but a number of new features have been added, the translation quality has been improved, and better internal processing methods have been introduced.

The processing improvements allow the Translator to be used on a minicomputer of modest size, at speeds comparable to that obtainable with DOTSYS on larger computers (about 1000 wpm on a Data General Corporation NOVA 800 with 64K bytes of memory). DOTSYS II and III, in turn, owe the fundamentals of their translation algorithm both to previous work in the field of braille translation by computer and to elementary concepts in the theory of automata, logic design, and formal languages. The background references and relationships discussed in the DOTSYS documents are relevant to the Duxbury Braille Translator also.

The source language used in coding the Translator is FORTRAN IV (UMF [74]). Nonstandard extensions to standard ANSI FORTRAN were isolated or avoided where possible. This should enable the Translator, with relatively little modification, to be run on any computer having a FORTRAN IV compiler and a modest amount of core storage (approximately equivalent to 43,000 bytes plus room for the operating system and its overhead).

The input text is presented to the Translator in the form of variable-length mixed-case lines (records), similar in appearance, when printed on a mixed-case printer, to ordinary typed text. These can be manually typed directly from inkprint or produced by another program (such as a compositor's tape converter) according to a fairly straightforward set of conventions. The output is the sequence of braille signs equivalent to the input text. Output can be produced in any of several forms, including "proof" output for a sighted editor and tactile (embossed) braille.

The Translator is almost completely table-driven; i.e., details of the translation algorithm are determined by tables read in at execution time rather than by the program itself. The Translator, as described in this document, comprises both the program and a standard set of tables. With modified tables, the program would be capable of processing different kinds of text, such as text containing mathematical or technical notation, languages other than English, or text containing nonstandard symbols for format control. (The present version has provision for "grade 1" handling of certain foreign languages, which is standard for foreign text within basically English-language works, and generally for literature used by foreign language students).

2 METHOD

GENERAL

The Translator comprises six cooperating processors: the basic reader (process B), the next-character handler (N), the translator (T), the stacker (S), the braille line composer (L), and the final output writer (O). It is the translator, as its name implies, that does most of the work of braille translation, and in fact, this section forms the main loop or program, the others being in the form of subprograms called by the translator to do their work at the appropriate time. However, in order to follow the processing of a given piece of text from inkprint form to braille form, it is easiest to imagine the processors running sequentially, each one in turn operating on the output, or a collection of outputs, from the previous processor.

The following descriptions of these processors are simplified to some extent, so that the discussion does not become mired in detail.

THE BASIC READER (B)

The B processor reads the input lines and produces a "stream" of individual characters in which most non-textual markers (such as the end-of-record) have been removed or are represented as a character. At this level, a hyphen at the end of a line is recognized as a signal to transmit neither the hyphen itself nor a special interline marker character that is otherwise given to the next (N) processor.

THE NEXT-CHARACTER HANDLER (N)

The N processor operates on the stream of characters produced by B and modifies it according to which of several modes it is in. For example, N may recognize a new paragraph condition, or insert tabulation or new-line controls for spaces or interline markers, or may delete the interline markers or substitute spaces for them. It also recognizes mode controls in the text that are pertinent to its operation, e.g. $COMPRESS will cause it to start compressing (deleting excess blanks in the data stream), $AS-IS to stop compressing.

Note that $COMPRESS, and practically all other "distinguished" symbols discussed in the text can be altered to some other symbol by changes in the tables, as detailed in Sullivan [76B].

THE TRANSLATOR (T)

The Buffer

The T processor operates on the stream of characters issued by N. As a first step, these are collected into a twelve-character sliding window, called the "buffer," so that a group of characters can be examined.

The Alphabet Table Lookup

The leftmost character in the buffer is looked up in the alphabet table. In this table, there is exactly one entry for each symbol that may appear in the text, containing, among other things: (a) a code denoting the braille sign for that symbol, (b) the symbol's "transition class," and (c) an index to that portion of the contraction table which pertains to this initial letter. Transition class is an arbitrary numerical code with purpose to be explained below.

The Contraction Table Search

The next step is to search that section of the contraction table indicated for this initial letter. In principle, this search may be visualized as a simple top-to-bottom entry-by-entry sequential search, although in fact a much faster "tree-search" algorithm is used. An entry in the contraction table consists of: (a) a string of up to nine characters (ten, counting the implied initial letter); (b) two "right-context class" designators; (c) an "input class" code; (d) a shift count; and (e) a set of up to four braille sign or control codes.

For a match to occur on a particular entry, three conditions must be satisfied. First, the entire string for that entry must correspond to the buffer, or left substring thereof. Secondly, the input class for the entry determines a set of "state variable" conditions that must be satisfied. A state variable is a logical switch, having a value "yes" or "no" according to its meaning and the text already translated. For example, a state variable whose meaning is "after a digit" would have the value "yes" if the character immediately preceding the one leftmost in the buffer had been one of the digits 0-9 and would have the value "no" in all other circumstances, including initially. "Meaning" is, strictly speaking, defined completely by entries in another table which governs the setting of these switches, called the transition table. This table will be discussed presently. The mechanism by which the input class selects a set of state variable conditions to be tested is called the decision table; this table is discussed in more detail under "The Decision Table," below.

The third condition for a contraction table match to occur is that the right-context classes be correct. The two characters immediately to the right of the matched string in the buffer are looked up in the alphabet table to determine whether they fall into classes indicated for this entry, e.g. right-context class "A"(for Alphabetic) would apply to any of the letters A - Z.

The contraction table search stops when a match occurs or the end of the table section for that particular initial letter is reached. In the latter event the shift count is taken to be 1, the transition class and braille sign code revert to those found for the initial letter, and processing proceeds. Otherwise, the shift, transition class (derivable from the input class via a table) and sign codes are taken from the matching entry.

The Buffer Shift

The buffer is then "shifted" left by the amount of the shift count, with new characters from the input stream entering on the right.

Braille Sign Output

The braille sign code(s) are sent to the stacker(S) constituting the output of the translator. These may directly represent braille signs, or they may be special control codes as is discussed in the section describing the stacker.

The State Transition

Finally, the transition class is used to determine a set of transitions to be applied to the state variables. For each variable, the transition may be specified as "no change," "toggle" (change "yes" to "no" and "no" to "yes"), "set to yes" or "set to no."

After the state variable transition, the process is repeated, beginning with the alphabet table search.

The Decision Table

The decision table determines whether the current settings of the state variables permit the use of a given contraction table entry. A particular decision table entry, corresponding to a "decision class," will demand that each state variable have a particular setting - "yes," "no," or "don't care"

(either setting). The entry is satisfied if all of the conditions (i.e. the logical conjunction thereof) are met by the actual current settings of the state variables.

This logic, though simpler and susceptible of more efficient implementation, is also inherently less general than the decision table logic of DOTSYS III, which realized a disjunction-of-conjunctions condition, possibly negated. However, this additional generality was never in fact used in the DOTSYS tables, and if it were needed with the present logic, it could be achieved in other ways -- namely, multiple contraction table entries.

THE STACKER (S)

The stacker operates upon the braille sign codes issued by the translator. In the simplest case, these codes are merely accumulated until the code for the blank braille sign is received -- i.e., a braille word is finished. This word, constituting one entry in a stack, is normally then removed from the stack and issued as output to the braille line composer. In exceptional circumstances, two words may be held in a stack before release. This is because the order may have to be changed; for example, in braille, units of measure are placed before the associated numeric quantity.

All special operations by the stacker, including delayed release and interchange of order, are directed by control codes issued by the translator. These are distinguishable from ordinary sign codes in that they have a value greater than 64.

THE BRAILLE LINE COMPOSER (L)

The line composer operates on braille words (stack entries) produced by the stacker. In each case, the length of the word will determine whether it can fit on the current braille line, an internal output buffer. If so, it is simply added thereto. Otherwise, the current line is sent to the final output writer, cleared, and started anew with the current braille word.

The line composer also concerns itself with counting lines to determine a new page condition, page numbering and titling, centering of headings, tabulation, special modes for poetry and hanging indentation and similar formatting matters. As with the stacker, all special operations of the line composer are controlled by control codes issued by the translator (and originating in the tables).

THE OUTPUT WRITER (O)

Braille lines composed by L as a sequence of sign codes are converted on a one-for-one basis to a suitable output code and written to the designated output file or device.

3 USAGE

OPERATION

This section describes usage under Data General's RDOS (RDOS [73]) operating system on a NOVA line computer, or an equivalent system. Operation on other systems may vary in some respects.

The general concept of operation is that overall control is exercised interactively at the main console ($TTO/$TTI), while all other input and output is to and from sequential files that, thanks to the generality afforded by RDOS, may be disk-or tape-resident or any device addressable by the system. The operator specifies the names of these files or devices at the console, together with a limited number of run parameters. Exception messages are directed back to the console during the progress of the run.

The translator is activated for a series of runs by the RDOS-level command

TRANSLATE

whereupon the translator announces itself, viz. DUXBURY BRAILLE TRANSLATOR 7/76 and prompts the operator for file names and run parameters as described in the next section. After the run, all files are closed and the translator asks:

MORE TRANSLATIONS?:

If the response is YES, the cycle resumes with the request for file names and parameters. Since one of the files is the driving table, which does not usually vary from run to run, provision is made to allow a given table to persist without re-reading. If the response to the "MORE" question is NO, a return to RDOS is made.

The translator has been equipped to permit a copy of the remarks (error and comment messages) it sends to the console to be directed to some other file or device. It requests the name of this file just before the self-annoucement message, with the prompt:

DESTINATION OF REMARKS?:

If such a copy is desired, a file name (as described for run parameters, below) should be supplied; otherwise respond "-1".

RUN PARAMETERS (CONSOLE DIALOG)

The following are the questions asked by the Translator of the operator. In all cases, response should be followed by a carriage return; as with RDOS console input in general, the rubout character (echoed as ←) may be used to delete one or more characters of a response before the carriage return. However, please note that the control-A character will interrupt back to RDOS.

In YES or NO responses, only the first two characters are checked. File names should be in standard RDOS form, e.g. XYZ, DPØ:ABC.D (using the "Ø" convention for zero.)

1. SOURCE OF TABLES?:
Respond with the name of a file containing binary tables to drive the Translator. For the standard tables, reply:

AMERICAN

The binary form of tables is generated by a table preparation program (Sullivan [76B]) User-generated tables may be run through this program for use with the Translator. The identity of the tables read in and the exact date of compilation is written back to the console for verification.

2. USE TABLES LAST READ IN?:
This question is always asked just after the tables are read in, to allow recovery in case the tables, as identified in the console message, are not the ones desired. The question is also the first one asked for the second and subsequent runs.
Respond YES or NO. If YES dialog proceeds to question 3, otherwise a return is made to question 1.

3. SOURCE OF TEXT?:
Respond with the file name of the input text to be processed. Preparation of the text file is described in later section.

4. CAPITALIZE AUTOMATICALLY?:
Respond YES if the input is in ordinary (mixed case) form and it is desirable to introduce braille capital signs in accordance with standard U.S. practice. Respond NO in either of the following two cases:
(a) Input text is in ordinary form and braille capitals are not wanted, in accordance with standard British and Canadian practice. (Note that it is possible to force occasional capitals in the text, if desired, with literal controls.)
(b) Input text is in old DOTSYS form, i.e. all uppercase with capitals entered literally. Note that in this case the literal capitals are not suppressed.

5. DESTINATION OF BRAILLE (EDITOR'S CODE)?:
Respond with the name of a file to receive the output, as described in a later section. The file may pre-exist or not. (In the former case, the old data is overwritten.)

6. LINES PER BRAILLE PAGE?:
Respond with an integer in the range 3-100 to specify the length of an output page in lines.

7. SIGNS PER BRAILLE LINE?:
Respond with an integer in the range 7 - 40 to specify the width of an output page in braille signs.

8. BRAILLE PAGES NUMBERED?:
Respond YES if it is desirable to number the braille output pages automatically, NO otherwise. If the answer is NO, questions 9 and 10 are not asked.

9. PG. NO. PARITY (1=ODD, 2=EVEN, 3=BOTH)?:
   Respond with an integer in the range 1 - 3 selecting which kinds of pages will be numbered. Note that, in accordance with standard practice, braille page 1 is never numbered.

10. STARTING BRAILLE PAGE NO.?:
    Respond with an integer in the range 1 - 999.

The Translator begins processing, issuing only exception messages (described in a later section) back to the console. After finishing the run, dialog resumes with question #2 as noted.

TABLE INPUT

As was noted above, table input is the "binary" (compacted) output produced by a preparation program. Another document (Sullivan [76B]) describes that program and the table formats.

TEXT INPUT

General Form

Text input is usually in the form of lines or records, up to 80 characters in length. The text is in most respects similar to ordinary typed material, i.e. there are both upper and lower case letters, paragraphs are indicated by skipped or indented lines, and tabular material may simply be aligned as desired without special format control indications. Such special controls are needed, however, for many of the special effects that may be desired in the braille; for example, headings to be centered must be surrounded by the symbols $HDS . . . $HDE (heading start and end).

Text of this kind may be produced by an editor such as the Duxbury Editor (Sullivan [76A]) or Data General's SUPEREDIT, and no doubt by many other means.

"Old" DOTSYS-style input, comprising fixed-length records all in upper case with true capitals indicated by a special flag, may also be processed if automatic capitalization is disabled (see "Run Parameters," above). Note, however, that many DOTSYS control symbols have been changed and the new ones must be used for correct operation.

Basic Text Handling Conventions and Modes

The most basic line-handling conventions are: (1) any blanks at the end of a line are always ignored and (2) a hyphen

at the end of a line is deleted and text on the next line treated as if it had immediately followed the character preceding the hyphen, with no intervening characters or markers. Thus "virtual lines" may be of any length, although "physical lines" are bound by the 80-character limit.

Beyond this, the ends of lines and spaces are treated differently depending on what combination of modes is selected. A mode is activated by including the appropriate symbol, e.g. $RUNNING (in either upper or lower case or mixed) surrounded by spaces, anywhere in the text itself. The selected mode then persists until changed by a mutually exclusive mode selection.

There are three mode groups, within which one and only one mode is active at a given time.

The first group pertains to automatic recognition of paragraphs. The initial and usual mode is $PAR. In this mode any skipped line or line beginning with two or more spaces is recognized as the beginning of a paragraph, and the program reacts exactly as if the symbol $P (described below) had been literally included. In the $NO-PAR mode, there is no special treatment of these situations. Note that when $PAR is in effect, the recognition of these special line-end situations takes precedence over the actions implied by the modes described below.

The second group relates to treatment of end-of-line. The usual and initial mode is $SPACING, in which the end-of-line is treated as a space. In $RUNNING mode, the end-of-line is completely ignored so that text is run together unless an actual space is entered at the beginning of each line. In $MARKING mode, the program behaves exactly as if the new-line symbol<(described below) had been entered at the end of each line. In this mode, in other words, original line endings will be preserved in the braille.

The third group relates to treatment of spaces. The initial and usual case is the $COMPRESS mode, in which multiple spaces are treated as one. In $SENTENCE mode, blanks are generally treated as in $COMPRESS mode except that up to two spaces are processed after a period, as was done in DOTSYS. In $AS-IS mode, all spaces are accepted and processed. Finally, in the $AUTO-TAB mode, all groups of spaces are processed as tabulations, i.e. exactly as if the tab symbol (described below) had been entered in the text. This mode permits entry of tabular material uncluttered by excessive control symbols.

Basic Typing Rules and Character Set

In general, subject to the form of text and prevailing modes as explained in the preceding sections, text is typed just as it appears in inkprint, using the same characters.

There are some exceptions to this general rule even in simple cases, and in certain cases more complex facilities must be called into play as will be explained in subsequent sections.

First, as was implied by the foregoing section, text may be typed free-form without regard to original line spacing, unless the special $MARKING effect is desired. A line should be skipped or a new line indented at least two spaces for each new paragraph.

The characters typed should ordinarily be the same as the inkprint in the following cases:

A - Z, a - z (letters)
0 - 9 (numbers)
: (colon)
; (semicolon)
. (period or decimal point)
, (comma)
- (hyphen - not when minus sign)
' (apostrophe - not when single quote)
" (double quote)
* (asterisk)
[] (brackets)
() (parentheses)
? (question mark)
! (exclamation point)

Single quotes should be typed as double quotes, even though a quotation may be included in another. In the case of quotes within quotes within quotes (or deeper), special rules apply as will be discussed. A minus sign also requires a special symbol, to be defined below.

A dash in inkprint should be typed as two hyphens (--), a long dash as four hyphens (----), without surrounding spaces. An ellipsis should be typed as three periods (...), without any intervening spaces.

The following special characters may be typed as they appear in inkprint only conditionally; if the conditions are not met a special symbol must be used:

$ (dollar sign) when followed by a number, e.g. $30.50.
& (ampersand) when surrounded by blanks, e.g. Boston & Maine.
% (percent sign) when not followed by a letter, e.g. 10%.
/ (slash) when followed by numbers or (unitalicized) letters, e.g. days/year.

The following characters have only special uses in text preparation, as will be described, and should not be copied

directly from inkprint. Many of these and other symbols that can appear in inkprint have special symbols defined for them (below); those that do not can also be spelled out, e.g. "equals" for "=":

= (equals sign)
_ (underscore)
@ (at sign)
\ (back-slash)
<> (angle brackets or inequality signs)
(number, pound or sharp-sign)
+ (plus sign)

The character ↑ sometimes represented as ^ may be typed in the input but it will be entirely ignored.

In the following sections, special situations not covered by these general rules are discussed.

Capitalization

Normally, capitalization is handled completely automatically by the program. However, when automatic capitalization is turned off by setting of a run parameter, any capitalized word to be represented as such in braille should be preceded by an equals sign (=) for initial cap, double equals sign (==) for all caps.

Italics

If only one, two, or three successive inkprint words are italicized, each of the words must be preceded immediately by an underline (_) when typed.

If four or more successive inkprint words are italicized, the first word must be preceded immediately by two underlines (_ _) and the last by one underline.

Accent Marks

Any accent or other diacritical mark used with a letter (such as é, è, ê, ä, ç) is represented by preceding the typed letter with a percent sign (%). For example, Abbé is typed Abb%e. However, see the section entitled "Foreign Languages" for proper handling of passages in foreign languages, other than anglicized words or proper names.

Ordering

When two or more special symbols must be typed (e.g. an italicized accented letter), possibly in conjunction with ordinary punctuation, the ordering should be:

Opening parenthesis or bracket
Opening quotation mark
Italic sign(s) (_ or__)
Letter sign (+, discussed below)
Accent sign (%)
Capital sign(s) (= or ==, if entered literally)

(Braillists will note that the last two are in the reverse of the correct braille order. The Translator will put them in correct order in the output, but requires this order of input for internal technical reasons.)

Replacement Symbols and Control Symbols in General

There are three classes of special symbols that may be included in text input to the Translator.

The first class is that of replacement symbols, i.e. groups of one or more characters that are (1) typed in place of an inkprint character or character sequence, or to express properties of the following character(s), and that (2) give rise to a sign or sign sequence in braille. (The percent, underscore and equals for accent, italics and capitals are thus replacement symbols.) An example would be &ae, entered for the ae diphthong in some instances of foreign-language text. The general typing rule for replacement symbols is that they be entered directly in place of the inkprint character, possibly abutting non-blank characters, with the alphabetic characters in lower case (unless automatic capitalization is disabled). Replacement symbols, excepting those already mentioned, always begin with the ampersand (&).

The second class includes all control symbols, i.e. those that cause some action, often relating to formatting, and usually without corresponding to some fixed sequence of braille signs. The general typing rule regarding these is that they be entered with surrounding spaces, although many control symbols would operate properly even if this restriction is not observed. Most control symbols begin with a dollar sign ($). Those that do not are < and >. Control symbols may be entered in either case (upper or lower or mixed). The symbols already introduced for basic text handling, e.g. $PAR, are examples of control symbols.

The third class comprises the editor's symbols; these will be discussed separately.

Special Character Replacement Symbols

Special input symbols must be typed to represent inkprint characters that are not in the character set listed above, or that cannot be typed directly under the restrictions mentioned:

Enter	to represent
&(lv)	long vowel sign (as in poetry or in foreign languages)
&(sv)	short vowel sign
&(ft)	end-of-foot sign (as in poetry)
&(cs)	caesura sign (as in poetry)
&($)	dollar sign
&(%)	percent sign
&(/)	slash
&(&)	ampersand
&(#)	number or pound sign
&(@)	at sign
&(+)	plus sign
&(-)	minus sign

In cases where numerous instances of such a special character occur in a passage, it is suggested that another non-occurring character (e.g. *) be typed instead of the special symbol, and that editing facilities be used later to systematically substitute the special symbol for the typed character.

Note that, due to the properties of braille, it will not always be appropriate to try to represent special symbols in situ. For example, because the period and dollar sign are the same sign in braille, an attempt to represent

```
He is worth million$ more...
```

by

```
He is worth million&($) more...
```

will come out in braille as

```
He is worth million.  more...
```

This is a concern of the editor, as will be discussed in a separate section.

Special Symbol Emphasis

In rare cases where it is necessary to emphasize that a literal symbol itself is important, as opposed to its meaning (e.g. in typing instruction manuals), the symbol

```
&(sym)
```

may be entered immediately before the symbol, whether a direct symbol or a replacement symbol. For example,

```
&(sym)%      for %
&(sym)&(@)   for @
&(sym)&(&)   for &
```

Forced Blanks

For special effects, it may be desirable to enter one or more blanks whose relationship to the surrounding text will be preserved in the braille (and not eliminated by $COMPRESS mode, for instance). A blank to be treated as if a non-blank braille character can be generated by entering

```
&b
```

once for each desired blank.

Quotation Marks

In cases of quotes within quotes within quotes, or deeper, the innermost quote marks should be entered specially, viz:

```
&(oq)          for opening quote marks
&(cq)          for closing quote marks
```

For this purpose, a citation of a letter is not considered a quotation, e.g.

```
..."i" comes before "e" ...
```

must be typed as-is, no matter how deep the clause may appear in levels of quotation.

Grade Switching

In some cases, it is necessary to translate text in a (nearly) letter-for-letter fashion, i.e. without allowing the contractions or many other rules of English Braille to be applied. An example would be certain foreign language passages, as will be discussed. Such a translation is called "Grade 1" braille; Standard Braille is "Grade 2."

Enter

$G1	to cause following text to be translated as grade 1.
$G2	to switch to grade 2
$G	to switch grades (from 1 to 2 or vice versa)

The use of $G, which has been retained from DOTSYS for compatibility reasons only, should be discouraged as it is not always obvious to a reader which way the switch is intended, and also because the effect of omitting a switch is to cause all subsequent text in the run to be translated in the wrong grade, even though other grade switches follow.

Foreign Languages

General Rules

Occasional foreign words and phrases in an English context must be translated in grade 1, but, except for Spanish, otherwise treated as English in regard to punctuation and the use of replacement symbols. The exception in regard to Spanish is that accented letters should use the special replacement

symbols for those letters (presented below), rather than the general accent (%) mark.

In cases of works incorporating substantial foreign language passages, as in English textbooks for the study of foreign languages, the foreign passages should be translated in grade 1 and special replacement symbols should always be used for accented letters or other characters peculiar to the language.

This statement of the rules is considerably simplified in relation to the Standard, and here as elsewhere the attention of an editor may be required.

Language Class and Grade Switches

The class of foreign language to be translated must be declared in advance by use of "$FL" (foreign language) mode switch. The two available are:

$FL-SPAN	Spanish
$FL-LIFG	Latin, Italian, French, and German

The mode switch need be entered only once in the text, unless the class is to change in the midst of the text, and may be placed anywhere before the foreign language passage(s). If no switch is present, $FL-LIFG is assumed.

A switch into grade 1 mode ($G1...$G2) should be made for the entirety of each foreign language passage.

Latin, Italian, French and German Symbols

For vowels with accents, enter "&" followed by the vowel followed by

- a for an acute accent (´)
- g for a grave accent (`)
- x for a circumflex accent (^)
- d for a diaresis or umlaut (¨)

For example, &ea is input for é. For other special symbols:

enter	for
&cc	ç (cedilla)
&ae	æ (diphthong)
&oe	œ (diphthong)
&(lv)	long vowel sign (before affected letter)
&(sv)	short vowel sign (before affected letter)

Spanish Symbols

For vowels with accents, enter "&" followed by the vowel for all acute (′) accents. Enter &d for ü. For other symbols:

enter	for
&n	ñ (n-tilde)
&?	¿ or ?
!	¡ or !
--	opening or closing conversation sign (--)

The conversation signs should be entered before the first quoted word or after the last with spacing conforming to ink-print.

Format Control Symbols

New Paragraph

Normally, new paragraphs are detected automatically in the input by the presence of skipped lines or a line indented two or more spaces. If this feature has been disabled (by $NO-PAR, described above), or if for any other reason it is convenient to indicate a new paragraph with a literal control, the symbol $P may be entered to indicate the beginning of a new paragraph. The text following $P is started on the next line in the braille output, indented two spaces (that is, starting in the third cell position).

New Line

The symbol $L may be used to ensure that following text starts on a new line. If it is encountered at a point where a new line would begin anyway, no line is skipped; consequently, multiple $L symbols will have no different effect from a single one. To skip a line unconditionally, or to skip multiple lines, see "Skip Multiple Lines" below.

Skip Multiple Lines

The symbol $SLn where n is a number (a non-negative integer) will skip the number of lines indicated, and output will continue from the left margin. For example, $SL37 will skip 37 lines; $SL1 will go to the next line; $SLØ will have no effect. No more than three braille pages will be skipped by this control.

The symbol

<

may also be used in place of $SL1.

New Page

The symbol $PG in the text will cause following text to be placed on a new page. The symbol $PGn, where n is an integer greater than or equal to 1, will cause the new braille page number to be set equal to n.

One-Time Tabulation

If the symbol $TABn is entered, where n is a positive integer, the text following the symbol will start at column n in the braille. Tabulation is implemented by the automatic insertion of spaces into the output and will therefore proceed to the next line if n is less than or equal to the present column number. Several variations, indicated by suffixes on the symbol, are also available for different types of tabulation.

The first set of variations has to do with the type of tabulation to be carried out. If a suffix R is attached, the tabulation will be such that the rightmost braille character of the word following the control symbol will fall into column n. This is to allow right-aligned columnar material. If a suffix D is attached, the tabulation is such that the decimal point, if any, in the following word (or the position beyond the rightmost character if there is no decimal point) falls into column n. This is to facilitate alignment of numerical tables. If a suffix C is attached, tabulation is such that the center character of the following word (or that just to the left of center if there are an even number of characters) falls into column n. As with ordinary tabulation, a new line would be used if the effect of tabulation would be to overwrite text. The suffix L, for left alignment, may also be attached; the interpretation is the same as the no-suffix case and so the L is not needed unless a filler suffix is also attached as described below.

The second set of variations permits "filler" characters to be inserted in the braille between the present position and the beginning of the tabulated text. A suffix "F" or "P" may follow the L, R, C or D suffix to indicate that filler characters are to fully occupy the intervening spaces or partially occupy them (blanks being preserved at either end). The character to be used as filler should be expressed in output (editor's) code and should follow the F or the P. The editor's code will be discussed presently, but assuming that the character 1 stands for the braille sign dot-5 (as it does in the generally used Snipas code), then an example of input might be

$L Parts $TAB29DP1 $23.26

In the braille, "Parts" would appear at the left of a new line, the decimal point of "$23.26" in column 29 of that line, and all but the two end spaces between would be occupied by the dot-5 sign.

Tabbed words are shifted left, if necessary, to avoid extending beyond the right margin. If an alignment code other than R, D or L is used, L is assumed. If a filler code other than F or P is used, F is assumed. In the present Translator, these corrective actions are taken without comment.

Tab Stops

Numbered tab stops can be set so that the user can right, left, center, or decimal point justify a word or number by stop number rather than by column. The general form of the control symbol for setting stops is $STBsan, where s is the stop number (1-9), a is the alignment type (R, L, C or D as explained under "One-Time Tabulation," above), and n is a column number. For example, the symbol $STB4L3 means set tab stop 4 to left justify on column 3. After a tab stop has been set, it may be used any number of times by inserting the symbol $# followed by the one digit number of the tab to be executed optionally followed by a 2-character filler suffix as explained above under "One-Time Tabulation." (Example: $STB4L3 $#4 LEFT -- will left-justify the word LEFT on column three. The symbol $#4 can be used any number of times.) If a tab is called for in a cell position less than or equal to the present position, output will begin on the next line.

The definition of a given tab stop number may be changed as often as desirable in the text. Initially, stops 1-8 are set to left-justify on column ((2 x stop number) + 1) (i.e. 3, 5, 7 etc.) and stop 9 to right-justify on column 40.

The symbol
>
may be used to cause tabbing to the next stop whose column number is greater than or equal to the current position in the output. No filler suffix is permitted. Thus > is equivalent to $#s where s is the stop number whose preset column is not less than the present column position but is less than the column of any other stop (tie goes to the lowest stop number). If no stop is set beyond the present position, > becomes equivalent to $FR, described below under "Flush-Right Tabulation."

Tab Clear

The control symbol $CTB "clears" all tab stop settings. More precisely, all stops are set to right-adjust at the far right of the page. (This is to facilitate removal of unwanted tab stops before setting up stops to be used automatically, i.e. in $AUTO-TAB mode.)

Flush-Right Tabulation

The symbol $FR, optionally followed by a two-character filler suffix as explained under "One-Time Tabulation," causes the following word to be aligned so that its rightmost character is full right on the page. For example, if the page width is set to 35, $FR is equivalent to $TAB35R.

Inkprint Page Number

The inkprint page number may be entered following the symbol $LEA (for "leaf") with an appropriate filler suffix as explained under "One-Time Tabulation," above. Generally, the suffix is "F" and the output code symbol corresponding to braille sign dots-3-6. In Snipas code this is the hyphen; thus the input to indicate inkprint page 36 would be

$LEAF- 36

Note that a space is required before the actual page number.

Hanging Indentation

The symbol $INDn, where n is a positive integer, sets the left margin, i.e. the first column that will be used in the braille output, to n until another $IND symbol is encountered. For example, $IND12 will have the first 11 columns blank, unless text is forced thereinto by tabulation, on subsequent lines. Initially, of course, $IND1 is assumed. (See also "Poetry, etc.", below)

Titles

Titles are placed between the symbols $TLS (Title START) and $TLE (Title END), (e.g. $TLS THIS IS A SAMPLE TITLE $TLE.) After a title has been inserted, each subsequent page has a centered title as its first line (the same line which contains the page number). If a new title is entered, it takes the place of the old on all future pages. Entering the title does not automatically turn to a new page.

Control symbols other than $TLS or $HDS (described below) may be generally used within titles, but tabulations will generally not have the desired effect because centering of lines is done after they are set up.

Titles may run to a maximum of two braille lines; thus at most one < symbol, for example, could appear in the input for a title.

Headings

These are similar to titles except that the control symbols are $HDS and $HDE and that headings are a one-time occurrence. The $HDS may be augmented by a nonnegative integer n, e.g. $hds1Ø, to cause a page eject before the first line of the heading unless at least n lines will be left on the current page after that first heading line. If no n is given, n=1 is assumed. ($hdsØ, which is equivalent to the DOTSYS $HDS, is permitted.) Whether or not a page eject occurs, headings are centered starting with a new line; they may run to any number of lines, each of which will be centered. After the $HDE, subsequent text will begin on a new line.

The same considerations with respect to control symbols within headings apply as for within titles, except for the limitation on the number of lines.

Poetry and Other Runover Indentation

The symbols $PTYS (Poetry START) and $PTYE (Poetry END) should be placed before and after all poetry text input. In this mode, the continuations of all poetry lines that exceed the physical length of the output line will be indented two spaces. The symbol $PTYS may be augmented to allow specification of the runover line indentation amount. The number of spaces to be left blank is specified by a number attached to the symbol, e.g. $PTYS4. (If no such number is given, 2 is assumed, as stated above.) Note that, unlike $tls... $tle and $hds ... $hde, $PTYS and $PTYE symbols need not necessarily occur in pairs. In fact, $PTYE is equivalent to $PTYSØ. Thus successive $PTYSn symbols may be used to vary the runover line indentation amount. This facility, together with the $INDn symbol (whereby all lines, i.e. the left margin, may be indented) permits different combinations of indentation for many purposes besides poetry.

Editor's Symbols

Role of the Editor

Occasionally, the involvement of a person familiar with braille will be necessary. This may be for special braille effects such as diagrams or tables presenting arrangement problems, or it may be to correct the braille output even though the input, prepared under the foregoing rules, be correct. The latter need, which may arise because of the approximate nature of the translation process, may be especially felt if "perfect" braille is required, as opposed to braille having some "misprint" toleration level.

The situations that give rise to braille translation problems for automatic processes are many and varied; the most serious are those deriving from the sound or meaning of the text. R.L. Haynes [68] of the American Printing House for the Blind remarks on this as follows:

"Some types of data encountered when translating inkprint into braille:

1. New words.
   Vietnamese
2. Variant spellings.
   greate (for greate) Conectecotte (for Connecticut)
3. Rarely used words.
   bioengineering salmonellosis
4. Letter sequences whose translation depends upon meaning.
   do (verb) do (musical note) said (verb) Said (place)
5. Compound words divided at the end of an inkprint line. Determination whether or not to use hyphen is based upon how word appears elsehere in the text.
6. Run together words.
   ' an Idon'tcareifittakesahundredyears attitude'
7. Foreign words. Foreign words are translated in Grade 1. A distinction must be made between foreign words which are names and those which are not. Names are put in Grade 2.
8. Acronyms. Translation depends upon whether or not the initials stand for separate words. SHARE SEATO DAR

9. Initials followed by periods. Spacing in braille may vary from inkprint depending upon whether or not the initials stand for a person's name.
   WASHINGTON D.C. D. C. JONES
10. Single letters. Sometimes a letter sign must be prefixed in braille.
   Ward C. C. Arnold A (article) big red A
11. Prepositions. Contraction in braille depends upon meaning.
   note 'to' in the phrase 'be friendly to all your'
   ...how wrong you can be about a person you have taken the trouble to be friendly to all your life, but at least...
   It is difficult to be friendly to all your neighbors.
12. Italics.
   a. Words italicized in inkprint are not always italicized in braille.
   b. Italicizing of a series of words in braille may be indicated in different ways depending upon whether or not the words are a title.
13. Measures. Abbreviations which follow numbers in inkprint precede the numbers in braille.
14. Hyphenated but not compound words. The hyphen is used in braille but may not end a line.
   say-ing th-th-them
15. Numbers separated by colons. Braille representation of time differs from representation of a reference.
   He came at 6:30. Genesis 3:12
16. Numbers separated by a hyphen. Usually the number sign is not repeated but there are exceptions.
   1956-58 5:10-5:20
17. Dash. A long dash in inkprint becomes a braille single dash if it is punctuation. A short dash in inkprint becomes a braille double dash if it represents an omission.
18. Blank lines. The effect of a blank line upon the format of the braille page depends somewhat upon nature of the preceding and following text material.
19. Chapter titles. Occasionally the number of lines required for a title in braille varies from the estimate based upon inkprint. This in turn may affect the ending of a page to begin the next chapter.

"The list above is not intended to be complete. The types of situations mentioned are not hypothetical but are based upon normal work in translation. All things considered, the application of data processing to braille translation has been successful. Correct

translation of the types of data mentioned above is achieved by a procedure which includes:

a. Occasional editing of inkprint copy by a braillist.
b. Insertion of some special control symbols by the keypunch operators.
c. Pre-translation reading of the text by the computer to locate a number of types of potential difficulties.
d. Scanning by a braillist of a test prooflisting of the format.
e. Proofreading of the braille text."

The Duxbury Translator handles many of the above situations correctly, some incorrectly. In particular, some known situations incorrectly handled are:

1. Plural citations of letters, e.g. Hs, p's, "q's". Single citations, with or without quotes, italics and accents, should be handled correctly (i.e. the letter sign used) in most cases.

2. Numbers followed by units of measure are correctly reversed and joined in most cases, but the number may end up on the following line if it cannot fit, which is incorrect. (also, as an example of a meaning-based problem, "feet" in A centipede has 100 feet...is taken to be a unit of measure.

The editor has three courses of action open to him. First, he can alter the output file directly, using any suitable editor program, before actual embossing. This is the electronic equivalent to manually eliminating or adding dots on previously prepared zinc plates; it is the easiest way to perform really small changes but not at all suitable for large changes, particularly where cascading effects are involved. Secondly, he can alter the Translator's driver tables, a process described in another document (see "Table Input," above). Modification of tables is usually indicated only when a word, acronym or other sequence is found to be mistranslated and it is expected that the sequence will be encountered again in subsequent works. Thirdly, he can enter special symbols in the input file, either the replacement and control symbols already discussed or the editor's symbols defined in the following sections.

Letter Sign and Number Sign

To insert a letter sign as is sometimes required by the braille rules to distinguish a character or sequence from a contraction or short-form word, insert the symbol

+

which acts like a replacement symbol. Note that the Translator correctly adds a letter sign in many instances.

Similarly, the Braille number sign may be inserted literally by the symbol

#

although the number sign is almost always inserted correctly and automatically by the Translator.

Division or Null Symbol - Preventing Contractions

When it is necessary to prevent the Translator from recognizing, i.e. grouping, a sequence of input characters, the symbol

//

may be entered within the sequence to prevent such recognition. This symbol acts as a replacement symbol, corresponding to no braille output signs.

The most common use of this symbol is to prevent contractions in unusual words where the Translator improperly contracts a sequence of letters. Many such cases involve strong syllable or root word boundaries within the sequence; correct translation can be achieved by placing a division symbol at the boundary, for example:

outhaul

would be incorrectly brailled with current tables (the "th" would be contracted) but entering the word as

out//haul

would force correct output.

Forcing Contractions

The use of the short form of any contractable letter group can be forced, regardless of context, by surrounding the letter group with the symbols "/_" and "_/". For example:

a/_dd_/

would cause the "dd" contraction to be used even though otherwise the program would not (and should not) use it at the end of a word. The 189 symbols of the form /_c_/ where c is a contractible letter-group, all act as replacement symbols. Note that if capitalized letters are present in c, then any capitalization must be explicitly entered (with =) prior to the /_.

Direct Braille

When the exact form of braille to be generated is known, the symbol

\

followed immediately by any number of characters in output (editor's) code, terminated by a space, may be used to generate the desired word in braille. The word thus composed is treated as any other word issued by the Translator, e.g. tabulation may be used to control its placement within a line.

Upper-case characters of the output code may be entered in lower case. Note that no replacement or control symbols will be recognized within the \... sequence; only the space terminates the symbol.

For example, a right-directed arrow, made up of several dots-2-5 signs and a dots-1-3-5 sign, may be entered as follows (assuming Snipas code): \:::::O

Termination Sign

In situations calling for the termination sign in braille, e.g. to terminate the effect of italics within a word, enter the symbol

@

which acts as a replacement symbol. Note that when automatic capitalization is in effect, cases of consecutive multiple capitals within a word may cause automatic introduction of termination signs as well as capital signs.

BRAILLE OUTPUT

Braille output is in the form of a file in braille-equivalent ASCII (American Standard Code for Information Interchange), usually made up of the 64 printable ASCII characters (not including lower-case letters) put into one-to-one relationship with the 64 braille signs. Such a code is called the output or editor's code and in some circumstances, as has been seen, the same code is used for input.

The coded lines and pages of this output file correspond to the lines and pages of braille. An embossing device (such as the Triformations, Inc. LED 120) can accept the file directly from the translator if it is "wired" for the output code and does not require special timing and control characters. However, it is usually desirable to store the output file on an electronic medium to permit perusal and editing as has been discussed, and to allow for multiple embossing runs without retranslations.

The output code is specified by the tables, which in standard form implement the Snipas code, defined in an appendix. Use of other codes should be discouraged, because of the implications for input compatibility with other installations.

EXCEPTION MESSAGES

Message remarking on unusual and perhaps erroneous situations arising during the translation process are directed back to the control console. These take the general form:

```
* ...  Message
NEAR IAL/C   a    b    OP/L   c    d    OAL   e
parameters
```

where:

1. The (0 or more) asterisks indicate by their number an estimate of the "severity level" of the condition, viz.:
   - 0 - Remark
   - 1 - Warning only (unusual but legal condition)
   - 2 - Error, probably fixed by automatic corrective action
   - 3 - Error, corrective action probably not sufficient for correct output
   - 4 - Error, corrective action impossible (program stops)
2. a and b are the absolute input line number and character number within the line at (or slightly before) which the condition occurred.
3. c and d are the braille page number and line number within page at or near which the condition occurred.
4. e is the absolute output line number at (or slightly beyond) which the condition occurred.
5. Parameters are additional information defined for a few messages.

The exception messages are as follows (severity level and parameter definitions are given in parentheses):

1. WORD OVERFLOW (3)
   A braille word is too long to fit on an output line. It is truncated on the right.

2. TITLE OVERFLOW (3)
   A title exceeds two lines in length. Lines between the first and last will be dropped.

3. CHAR. OUTSIDE RANGE: (3, decimal code of character in question)
   A character not in the allowed range of ASCII input characters has been encountered. It is ignored.

4. INCORRECT SIGN CODE (2, code number)
   An error, probably in the Translator or tables, has caused an invalid braille sign or control code to reach the output processor. It is processed as a blank.

5. BRAILLE STACK FULL (2)
   An error, probably in the tables, has caused too many stack elements to be requested in the S processor. The stack is emptied.

6. ERRONEOUS SIGN OR CONTROL CODE (3, code number)
   This has a meaning similar to #4, with respect to the stacker (S processor).

7. UNDEF. CHAR.: (3 decimal code of character in question)
   A character has been encountered that is not in the table-defined input character set. (This is a finer test than for message #3.) The character is processed as an asterisk (*).

8. EXCESS CHARACTER IGNORED: (3, decimal code of character in question). An excessively long input word (group of contiguous nonblank characters) has been encountered. Each character beyond the 40th gives rise to this message but is otherwise ignored.

9. OP c NEAR IAL a (ø)
Processing has started on the output for braille page c, where c is divisible by 1ø. The input line number is approximately a. (No NEAR IAL/C ... line is produced with this message.) This message is to keep the console operator informed of progress and also to provide a braille - vs. -input cross-reference for later editing purposes.

APPENDIX A SNIPAS CODE

The braille-equivalent editor's code defined below is the work of the late Richard J. Snipas, with some suggestions from others. It sets up a 1-1 equivalence (substitution cipher) between the 64 braille cells and the 64 printable upper-case ASCII characters, based on mnemonics.

The explanations given in the tables are for the most part in Mr. Snipas' own words.

The correspondence between editor's codes such as the Snipas code and braille should not be confused with the correspondence between inkprint (translator input) and braille. For one thing, the latter may not be one-for-one. As an example. the input sequence

1 world

would (presuming English rules) be translated into the braille sequence

i.e.

dots 3-4-5-6, 1, space, 4-5-6,2-4-5-6

which would be represented (assuming Snipas Code) by the ASCII code sequence

#A 1W

Note particularly the quite different (and unrelated) roles of the "1" in inkprint and in Snipas Code.

Character	Braille Dot Equivalent
A through Z	
0 (zero)	4, 5
1	4, 5, 6
2	5, 6
3	1, 4, 5, 6
4	4
5	5
6	6
7	2, 3, 5, 6
8	1, 2, 6
9	1, 4, 6
! (Exclamation Mark)	2, 3, 5
" (Quotation Mark)	3, 5, 6
# (Number Sign)	3, 4, 5, 6
$ (Dollar Sign)	2, 5, 6
% (Percent Sign)	1, 2, 4, 5, 6
& (And Sign)	1, 2, 3, 4, 6
' (Apostrophe)	3
((Left Parenthesis)	1, 2, 3, 5, 6

Explanation
The letters A through Z are standard braille configurations
Seldom used dot configuration -- saves confusion
Looks like print character
Looks like Braille 2
Braille "TH" sign -- THREE
Print 4 is dot 4
Print 5 is dot 5
Print 6 is dot 6
Nemeth code number 7
Braille "GH" sign -- EIGHT
Looks like print 9
Standard Braille
Standard Braille Right quote mark
Standard Braille
Standard Braille
Braille "ER" sign -- PERCENT
Grade 2 Braille "AND" Sign
Standard Braille
Nemeth code

Character	Braille Dot Equivalent
) (Right Parenthesis)	2, 3, 4, 5, 6
* (Asterisk)	1, 6
+ (Plus Sign)	3, 4, 6
, (Comma)	2
- (Hyphen)	3, 6
. (Period or Decimal)	4, 6
/ (Slash)	3, 4
: (Colon)	2, 5
; (Semi-colon)	2, 3
← (Left Arrow)	2, 4, 6
= (Equal Sign)	1, 2, 3, 4, 5, 6
↑ (Up Arrow)	3, 4, 5
? (Question Mark)	2, 3, 6
@ (At Sign)	1, 5, 6
[(Left Bracket)	2, 3, 4, 6
] (Right Bracket)	1, 2, 4, 6
\ (Backward Slash)	1, 2, 5, 6
> (Greater Than)	3, 5
< (Less Than)	2, 6
(Space)	No dots

Explanation
Nemeth code
Nemeth code
Nemeth code
Standard Braille
Standard Braille
Standard Braille Decimal Point
Standard Braille
Standard Braille
Standard Braille
Looks like print
Nemeth code
Braille "AR" Sign -- ARROW
Standard Braille
Braille "WH" Sign -- WHITCH's HAT (sorry for the bad spelling)
Braille "THE" Sign and "ED" Sign -- Left Bracket, plus Right Bracket spells "THEED"
No special reason for this one
Looks like bottom of Greater Than Sign.
Looks like bottom of Less Than Sign

APPENDIX B SPECIAL SYMBOLS INDEX

REFERENCES

EBAE(72) English Braille, American Edition, 1959 (Revised 1972), American Printing House for the Blind, Louisville, Kentucky.

Haynes(68) R. L. Haynes, "Computer Translation of Grade II Braille", Proceedings Conference on New Processes for Braille Manufacture, 1968, American Printing House for the Blind, Louisville, Kentucky, pp. 1-4.

Millen(70A) J. K. Millen, DOTSYS II: Finite-State Syntax-Directed Braille Translation, MTR-1829, The MITRE Corporation, (1970).

Millen(70B) J. K. Millen, DOTSYS II: User's Guide and Transfer and Maintenance Manual, MTR-1853, The MITRE Corporation, (1970).

RDOS(73) RDOS Real-Time Disk Operating System User's Manual Data General document 093-000075-04.

Sullivan(75) J. E. Sullivan (ed.), DOTSYS III: A Portable Program for Braille Translation, MTR-2119 (Rev. 1), The MITRE Corporation (October 1975).

Sullivan(76A) J. E. Sullivan, The Duxbury Editor User's Manual, Duxbury Systems, Inc. document DS-JES-760704 (July 1976).

Sullivan(76B) J. E. Sullivan, The Duxbury Braille Table Preparation Program, Duxbury Systems, Inc. document DS-JES-760718 (July 1976).

UMF(74) User's Manual, FORTRAN IV. Data General Corporation document no. 093-000053 (July 1974).

ACKNOWLEDGEMENTS

This document was derived partly by copying the current DOTSYS III description (Sullivan[75]), with extensive deletions, additions and changes as necessary. The portions of that original document contributed by Dr. J. K. Millen and W. R. Gerhart of the MITRE Corporation, pertaining as they did to the original COBOL implementation and to editorial procedures that have been altered, are not included in the present description; but obviously their original work, especially the basic algorithm design of J. K. Millen, and the painstaking table work of both of them, remains an important influence on the development of the system.

The continuing interest and contribution of R. A. J. Gildea, original leader of the DOTSYS project and now consultant to Duxbury Systems, Inc., and S. M. Simpson, President of DS, owner of the facility on which the system was developed and knowledgeable contributor to the programming and myriad other matters, are also acknowledged. Finally, the benefit of ideas from numerous other persons, especially those who have used or modified DOTSYS, and those who have expressed early interest in the Duxbury Translator and have influenced its design, is also recognized.

J. Sullivan

Duxbury Systems

Post Office Box 1523
Duxbury, Massachusetts 02332
Telephone: 617-934-6627

July 23, 1976
Technical Memorandum
Doc. #DS-JES-760723

To: General Distribution
From: J. E. Sullivan
Subject: Annotated example of EPROOF operation

EPROOF is a program to display the result of a Duxbury Braille Translator run in a "proof" (printed dot) format. Its operation is explained by the example of console dialog below. The RDOS command EPROOF initiates the program.

```
DUXBURY BRAILLE PROOF PROGRAM 7/76
SOURCE OF TABLES?: AMERICAN
TABLES:
DUXSYS 7/76              07/19/76 08:32:20
USE TABLES LAST READ IN?: YES
SOURCE OF EDITOR'S CODE?: TXEXERB
DESTINATION OF PROOF BRAILLE?: $LPT
FOR DOTS - WHICH ONE OF (.? *? 0?)?: 1
EDITOR'S CODE?: YES
EXTRA SPACE?: YES
MORE EMBOSSING?: YES
USE TABLES LAST READ IN?: YES
SOURCE OF EDITOR'S CODE?: TXSNIPASB
DESTINATION OF PROOF BRAILLE?: $LPT
FOR DOTS - WHICH ONE OF (.? *? 0?)?: 1
EDITOR'S CODE?: YES
EXTRA SPACE?: YES
MORE EMBOSSING?: NO
STOP

R
```

Name of the Tables used in the Translator run.

If these are the correct tables.

Name of file (or device) containing Translator output.

File or device to recieve printed braille.

Selection of dot representation.

If editor's code is to be printed below the printed braille signs.

For extra spacing between the printed braille lines.

To recycle for more embossing. Remaining responses are as above.

J. Sullivan

THE DUXBURY BRAILLE TABLE PREPARATION PROGRAM
USER'S GUIDE
DS-JES-760718-00 01

Ordering No. DS-JES-760718

P. O. Box 1523
Duxbury, Massachusetts 02332

NOTICE

Original Release: July, 1976
Revised: October, 1976

Written for Duxbury Systems, Inc. by Joseph F. Sullivan

PREFACE

This document is intended to supplement the user's guide for the Duxbury Braille Translator (Sullivan 76), and explains how the separate Duxbury Braille Table Preparation Program may be used to alter the tables that drive the Translator. A general explanation of how the tables are used is contained in the Translator document.

This description corresponds to the Duxbury Braille Table Preparation Program of 7/76.

The appendices correspond to the tables DUXSYS 7/76, release 8/9/76.

TABLE OF CONTENTS

INTRODUCTION

PURPOSE

The Duxbury Braille Table Preparation Program may be used to generate new versions of the tables that drive the Duxbury Braille Translator, as described by Sullivan [76]. The tables are supplied with the Duxbury Translator and form an integral part of the process described by the cited Translator document. However, circumstances may arise such that the user may wish to alter or augment the tables. One such circumstance might be a word, found to be incorrectly translated, that occurs so often in several texts that it is impractical to force the correct translation by special treatment (for example, by the "//", "/__", and "_/" control symbols). In such a case, an addition to the contraction table is called for.

TABLE VERSION CONTROL

In accordance with the practice for DOTSYS III tables (Sullivan [75]), since the Duxbury Translator tables are a variant of these in both form and content, if changes are made to a table that is shared by other institutions, a new version has been created and it should be labeled according to the authoring institution and date, e.g., "CNIB 11/76." Subsequent alterations at the same institution, prior to any external distribution, need not be considered new versions.

Because table changes are an intricate process, it is recommended that a detailed journal be kept of all such changes and the reasons therefor.

OPERATION -- CONSOLE DIALOG

The RDOS command PREPARE starts the preparation program. The following dialog illustrates and explains the dialog that then takes place at the main console. Note that the program is a simple "command interpreter" -- thus more than one table can be processed by one activation, and more than one copy of a processed table can be produced (by giving several "writes" before another "read" or "stop"). All "files" may be either named disk files or device names.

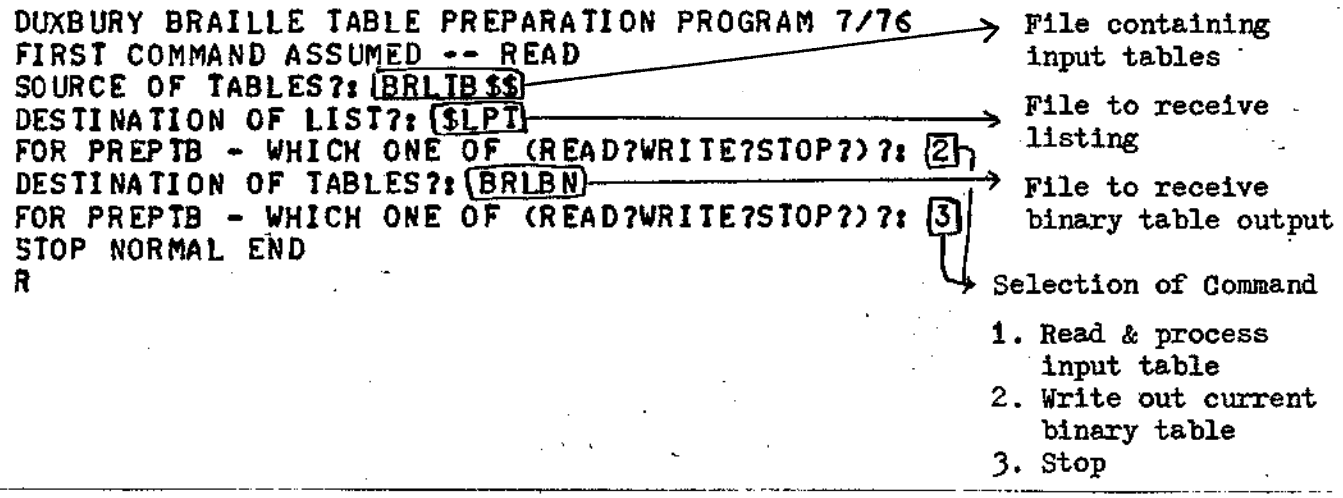


TABLE INPUT

Function and Ordering of Certain Tables

The Alphabet Table

The alphabet table identifies all legal text input characters. The order of input is arbitrary, but is related to that of the contraction table (see below).

The Contraction Table

The contraction table contains not only contractions but many other sequences related to the heuristics of the translation process, and the definition of special control symbols.

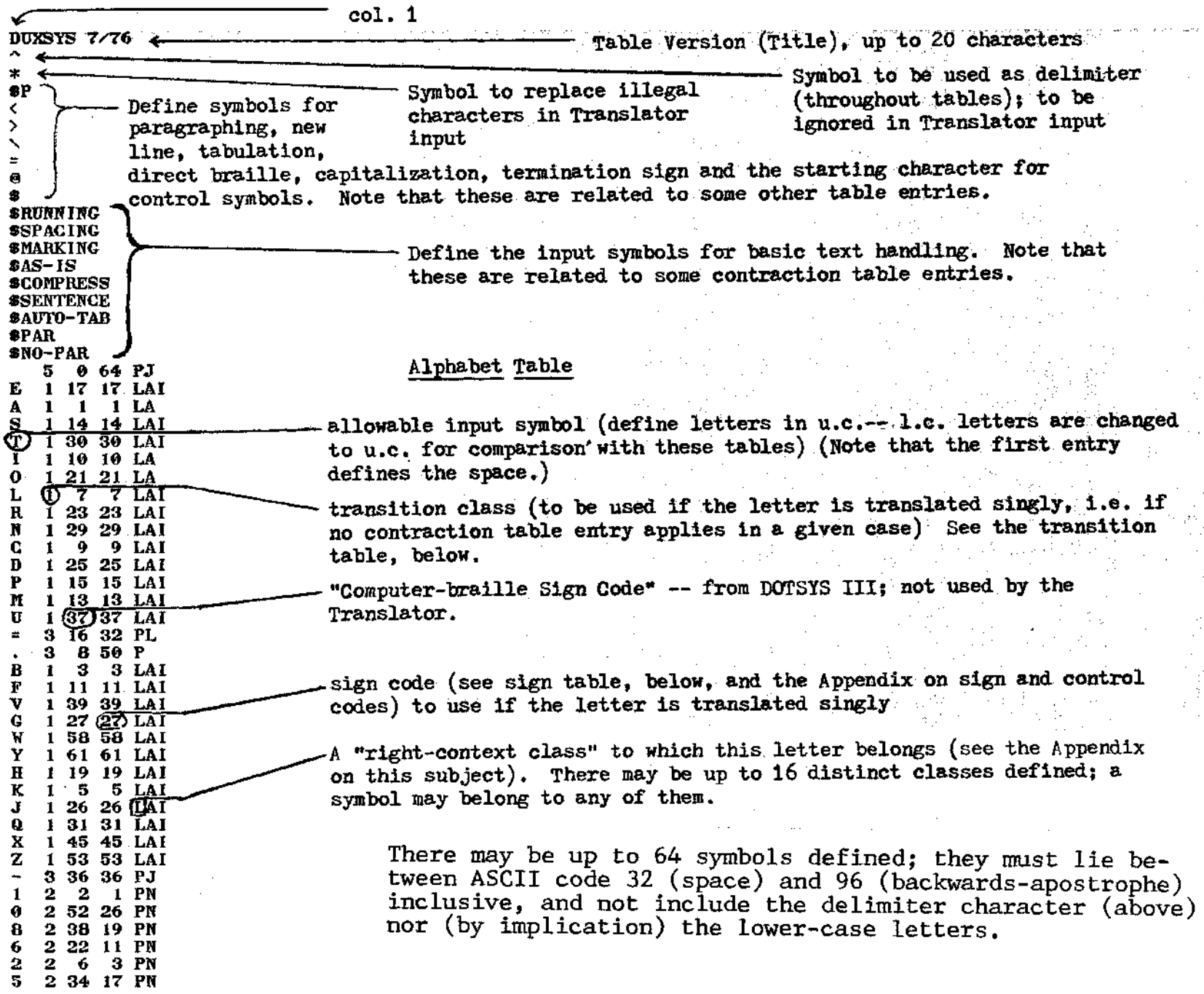
Entries in the contraction table beginning with the same symbol must be grouped together, and these groups must be arranged in the same order as the corresponding symbols in the alphabet table. Also symbols in the alphabet table that have no corresponding section in the contraction table are grouped, in any order, at the end of the alphabet table.

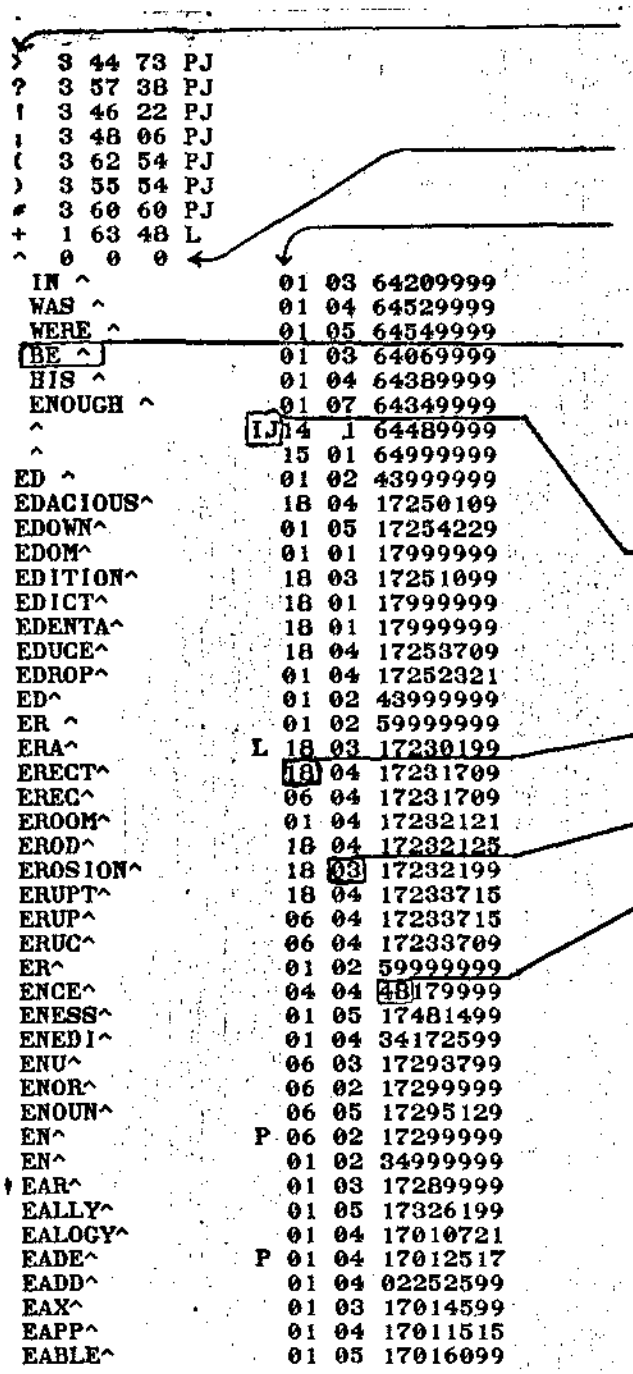
Within a section of the contraction table determined by a

common initial letter, all entries having a given second character must also be grouped together. The order of input of these subsections is completely arbitrary and usually will vary from section to section as a function of conditional frequency. This grouping scheme is continued right up to the 10th character. In general, if two entries agree through the nth character, then all entries between them must also agree with them through the nth character.

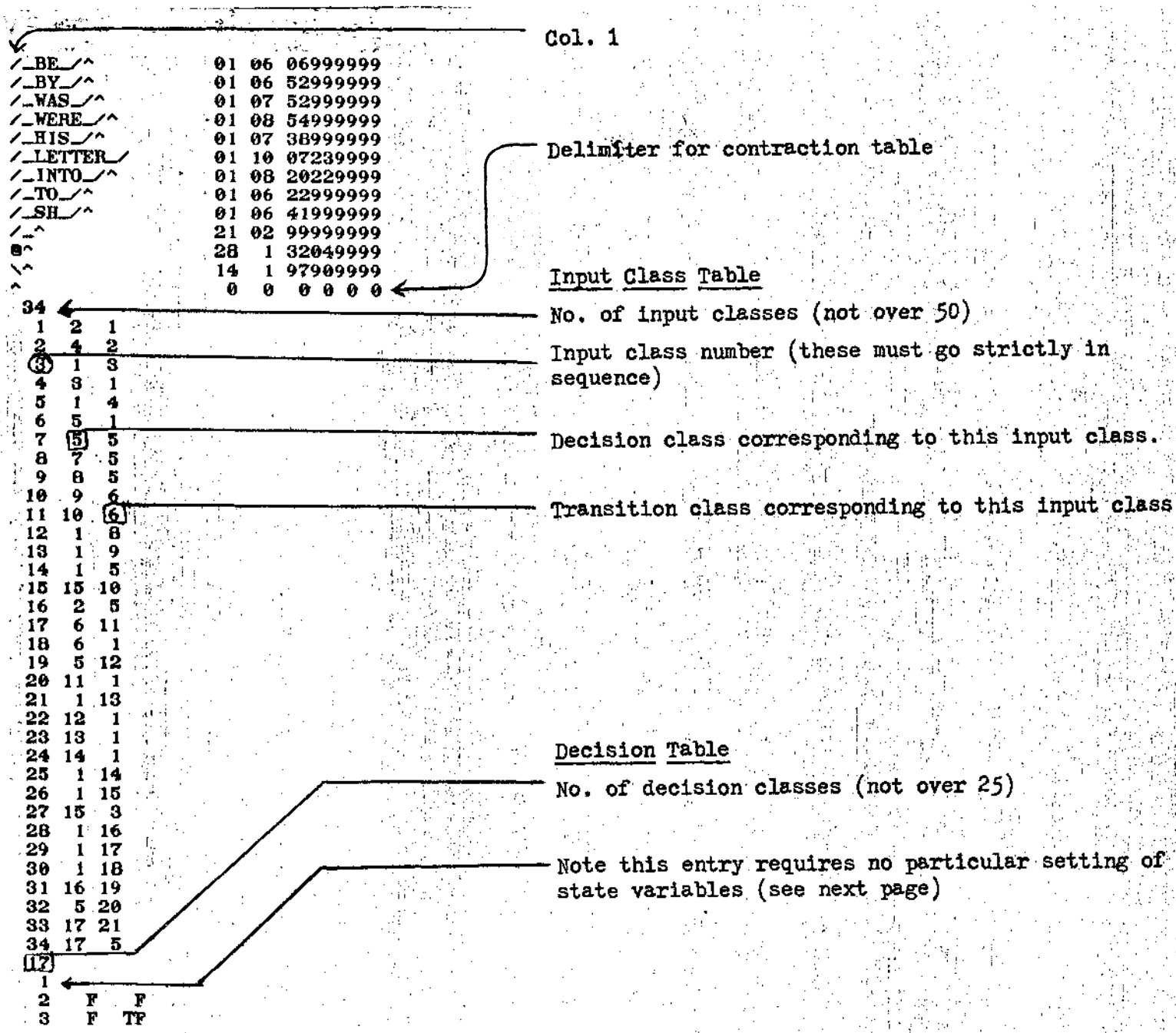
Table Input Specifications

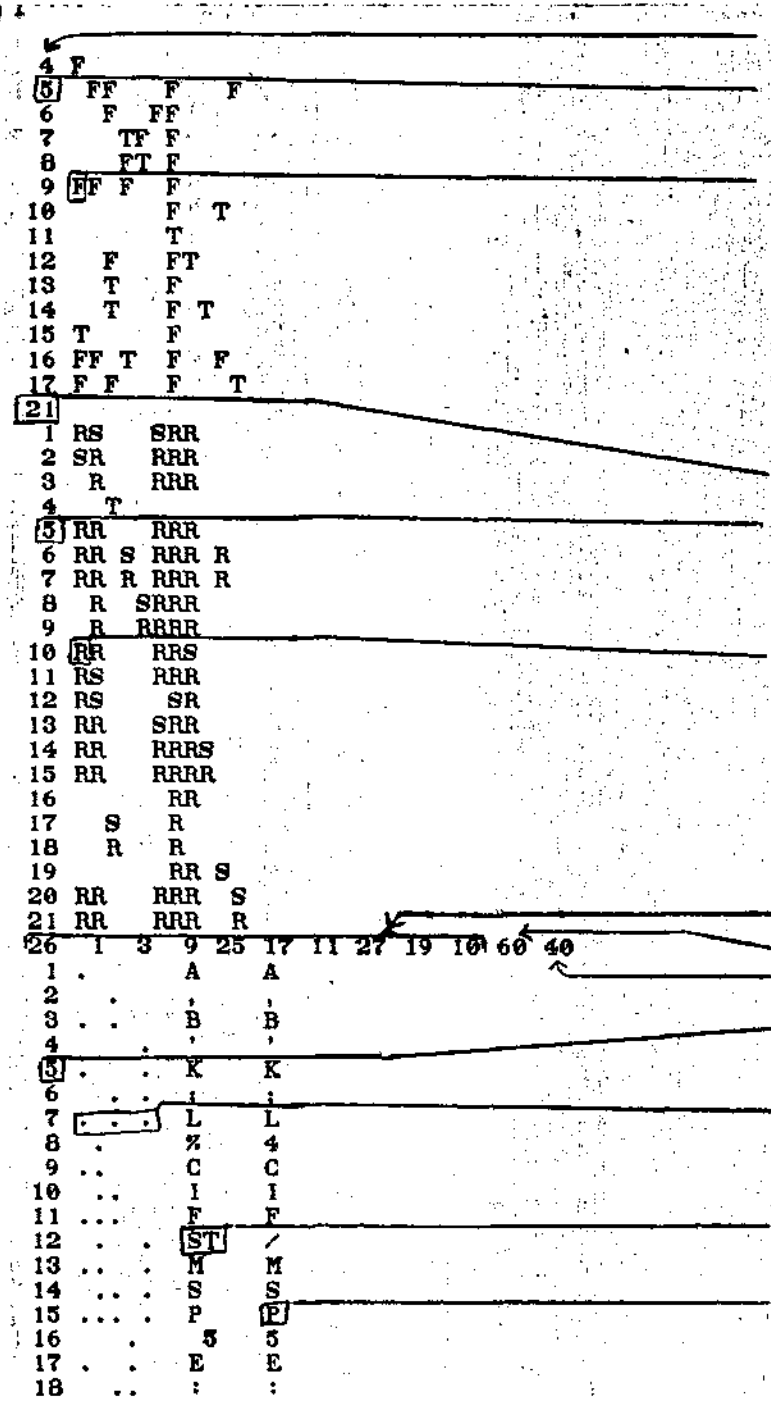
The following pages define, by means of an explained example, the format of the input tables to the Preparation Program. Note that the illustration does not include the middle of the alphabet table, the middle of the contraction table nor the end of the sign table. Note also that this illustration does not necessarily correspond in the parts shown to the tables of 7/76 as distributed.


col. 1
DUXSYS 7/76
Table Version (Title), up to 20 characters
^
*
Symbol to be used as delimiter (throughout tables); to be ignored in Translator input
$P
<
>
/
=
@
$
Symbol to replace illegal characters in Translator input
Define symbols for paragraphing, new line, tabulation, direct braille, capitalization, termination sign and the starting character for control symbols. Note that these are related to some other table entries.
$RUNNING
$SPACING
$MARKING
$AS-IS
$COMPRESS
$SENTENCE
$AUTO-TAB
$PAR
$NO-PAR
Define the input symbols for basic text handling. Note that these are related to some contraction table entries.
Alphabet Table
5 0 64 PJ
E 1 17 17 LAI
A 1 1 1 LA
S 1 14 14 LAI
T 1 30 30 LAI
I 1 10 10 LA
O 1 21 21 LA
L 1 7 7 LAI
R 1 23 23 LAI
N 1 29 29 LAI
C 1 9 9 LAI
D 1 25 25 LAI
P 1 15 15 LAI
M 1 13 13 LAI
U 1 37 37 LAI
= 3 16 32 PL
. 3 8 50 P
B 1 3 3 LAI
F 1 11 11 LAI
V 1 39 39 LAI
G 1 27 27 LAI
W 1 58 58 LAI
Y 1 61 61 LAI
H 1 19 19 LAI
K 1 5 5 LAI
J 1 26 26 LAI
Q 1 31 31 LAI
X 1 45 45 LAI
Z 1 53 53 LAI
- 3 36 36 PJ
1 2 2 1 PN
0 2 52 26 PN
8 2 38 19 PN
6 2 22 11 PN
2 2 6 3 PN
5 2 34 17 PN
allowable input symbol (define letters in u.c.-- l.c. letters are changed to u.c. for comparison with these tables) (Note that the first entry defines the space.)
transition class (to be used if the letter is translated singly, i.e. if no contraction table entry applies in a given case) See the transition table, below.
"Computer-braille Sign Code" -- from DOTSYS III; not used by the Translator.
sign code (see sign table, below, and the Appendix on sign and control codes) to use if the letter is translated singly
A "right-context class" to which this letter belongs (see the Appendix on this subject). There may be up to 16 distinct classes defined; a symbol may belong to any of them.
There may be up to 64 symbols defined; they must lie between ASCII code 32 (space) and 96 (backwards-apostrophe) inclusive, and not include the delimiter character (above) nor (by implication) the lower-case letters.




```
>   3 44 73 PJ
?   3 57 38 PJ
!   3 46 22 PJ
;   3 48 06 PJ
(   3 62 54 PJ
)   3 55 54 PJ
#   3 60 60 PJ
+   1 63 48 L
^   0  0  0
 IN ^              01 03 64209999
 WAS ^             01 04 64529999
 WERE ^            01 05 64549999
 BE ^              01 03 64069999
 BIS ^             01 04 64389999
 ENOUGH ^          01 07 64349999
 ^              IJ 14  1 64489999
 ^                 15 01 64999999
ED ^               01 02 43999999
EDACIOUS^          18 04 17250109
EDOWN^             01 05 17254229
EDOM^              01 01 17999999
EDITION^           18 03 17251099
EDICT^             18 01 17999999
EDENTA^            18 01 17999999
EDUCE^             18 04 17253709
EDROP^             01 04 17252321
ED^                01 02 43999999
ER ^               01 02 59999999
ERA^            L  18 03 17230199
ERECT^             18 04 17231709
EREC^              06 04 17231709
EROOM^             01 04 17232121
EROD^              18 04 17232125
EROSION^           18 03 17232199
ERUPT^             18 04 17233715
ERUP^              06 04 17233715
ERUC^              06 04 17233709
ER^                01 02 59999999
ENCE^              04 04 48179999
ENESS^             01 05 17481499
ENEDI^             01 04 34172599
ENU^               06 03 17293799
ENOR^              06 02 17299999
ENOUN^             06 05 17295129
EN^             P  06 02 17299999
EN^                01 02 34999999
EAR^               01 03 17289999
EALLY^             01 05 17326199
EALOGY^            01 04 17010721
EADE^           P  01 04 17012517
EADD^              01 04 02252599
EAX^               01 03 17014599
EAPP^              01 04 17011515
EABLE^             01 05 17016099
```


Col. 1
/_BE_/^ 01 06 06999999
/_BY_/^ 01 06 52999999
/_WAS_/^ 01 07 52999999
/_WERE_/^ 01 08 54999999
/_HIS_/^ 01 07 38999999
/_LETTER_/ 01 10 07239999
/_INTO_/^ 01 08 20229999
/_TO_/^ 01 06 22999999
/_SH_/^ 01 06 41999999
/_^ 21 02 99999999
@^ 28 1 32049999
/^ 14 1 97909999
^ 0 0 0 0 0 0
Delimiter for contraction table
Input Class Table
34
No. of input classes (not over 50)
1 2 1
2 4 2
3 1 3
4 3 1
5 1 4
6 5 1
7 5 5
8 7 5
9 8 5
10 9 6
11 10 6
12 1 8
13 1 9
14 1 5
15 15 10
16 2 5
17 6 11
18 6 1
19 5 12
20 11 1
21 1 13
22 12 1
23 13 1
24 14 1
25 1 14
26 1 15
27 15 3
28 1 16
29 1 17
30 1 18
31 16 19
32 5 20
33 17 21
34 17 5
Input class number (these must go strictly in sequence)
Decision class corresponding to this input class.
Transition class corresponding to this input class.
Decision Table
No. of decision classes (not over 25)
12
1
2 F F
3 F TF
Note this entry requires no particular setting of state variables (see next page)



col. 3

Decision class number (these must go in sequence)

A state variable condition. This one (in col. 5) applies to state variable 1 (see the appendix) and demands that it be "false" or "off." T is used for "must be true" and a blank for "don't care." For example, this entry requires 1, 2, 4 and 7 to be "false." Up to 16 state variables may be referred to in this and the transition table.

Transition Table

No. of transition classes (no more than 25).

Transition class number (these must go in sequence).

A transition definition. This one (in col. 5) is for state variable 1. The code is

R	reset (turn to "false" or "off")
S	set (turn to "true" or "on")
T	toggle ("change," i.e. logical "not")
blank	no change

Sign Table

Sign codes for the digits 0-9

Sign code for the number sign

Sign code for the decimal point

Sign code being defined. These must go exactly 1-64 in sequence.

Dots for this sign, starting in col. 5 in the order 1, 4, 2, 5, 3, 6. (Used for Preparation Program display only, no significance to Translator.)

"Mnemonic" for this code, up to 3 characters. (Also used only for Preparation Program display.)

Output (editor's) code corresponding to this sign. As described in the Translator document, this symbol is also used in input under some circumstances.

LISTED OUTPUT

The output listing is for the most part an echo of the input, with some adjustment of the spacing and headings added. An extra section is the "Compiled Alphabet and Contraction Tables," a merged listing of these two tables as illustrated and annotated on the next page.

A comment will be printed at the end of this list, giving the total number of entries in the contraction table and the number allowed, and also the space used by the table (in bytes) and the number allocated. (The limits are applied jointly; i.e. neither can be exceeded.) The space used is a complex function of the entries. An entry will generally take less space if it

(1) has fewer symbols in the recognition string;
(2) has fewer right context checks;
(3) uses input class 1;
(4) shifts 2 characters of input; and
(5) has fewer sign codes to output.

```
/_TO_/^          1  6 22999999
/_SH_/^          1  6 41999999
/_^             21  2 99999999
@^              28  1 32 49999
\^              14  1 97909999
^                0  0  0 0 0 0

COMPILED ALPHABET AND CONTRACTION TABLES:

AN = ALPHABET ENTRY NO.
S  =   SYMBOL
TC =   TRANSITION CLASS

SG =   BRAILLE SIGN
CS =   COMPUTER BRAILLE SIGN
RC =   RIGHT-CONTEXT CLASSES (NUMERIC)
CN = CONTRACTION ENTRY NO.
ST =   STRING
RC =   RIGHT-CONTEXT CLASSES
IC =   INPUT CLASS
SH =   SHIFT AMOUNT
SG =   BRAILLE SIGNS

 AN S TC SG CS RC

      CN ST          RC       IC SH SG

 1    5 64  0 0 1

       1 IN ^        255 255   1  3  64 20 99 99     2
       2 WAS ^       255 255   1  4  64 52 99 99     4
       3 WERE ^      255 255   1  5  64 54 99 99    -1
       4 BE ^        255 255   1  3  64  6 99 99     5
       5 HIS ^       255 255   1  4  64 38 99 99     6
       6 ENOUGH ^    255 255   1  7  64 34 99 99     7
       7 ^             4   1  14  1  64 48 99 99     8
       8 ^           255 255  15  1  64 99 99 99     0

 2 E  1 17 17 2 3 4

       9 D ^         255 255   1  2  43 99 99 99    19
      10 DACIOUS^    255 255  18  4  17 25  1  9    11
      11 DOWN^       255 255   1  5  17 25 42 29    13
      12 DOM^        255 255   1  1  17 99 99 99    -2
      13 DITION^     255 255  18  3  17 25 10 99    15
      14 DICT^       255 255  18  1  17 99 99 99    -2
      15 DENTA^      255 255  18  1  17 99 99 99    16
```

Heading Explanations

Alphabet table headings

Contraction table headings

Right-context classes (numeric) (255=none)

Alphabet entry (note symbol is blank)

"Branch" -- added automatically to speed searching in the Translator.

Contraction Entries

BINARY TABLE OUTPUT

This file is in the tables in an internal, compacted form required for input to the Translator.

EXCEPTION MESSAGES

The general-purpose comment

** TABLE SEQUENCE ERROR **

appearing in the listed output means that a violation of one of the input ordering or size restrictions has been detected. The binary table generated in such cases will be invalid and should not be used with the Translator.

In the special case of the contraction table "space" capacity being exceeded, the comments

* * * * CONTRACTION TABLE CAPACITY EXCEEDED
STOP JSTO

will appear on the console, and the run will stop abnormally.

APPENDIX A Braille Sign and Control Codes

Code	Meaning
00	required blank
01-63	braille sign codes
64	end of braille word (blank or end of line)
65	new line (if not already on a new line)
66	unused (reserved for: line composition function)
67	paragraph start
68	one-time tab (skip to col. nn)
69	new page
70	new line (unconditional)
71	skip (multiple lines)
72	tab (skip according to permanent tab)
73	tab to next stop
74	"flush right" tab
75	set left margin
76	clear tabs
77-80	unused (reserved for: line composition function)
81	start heading input
82	end heading input
83	unused
84	set continuous text stacking mode
85	idle (reserved for: reset continuous text stacking mode)
86	unit of measure (reverse stack)
87	unused
88	start running title input
89	end of running title input
90	accept input text directly until blank
91-92	idles (reserved for: self-checking mode)
93	set tab stop
94	idle (reserved for: octal braille)
95	start poetry mode
96	end poetry mode
97	start direct braille
98	end of run
99	(filler in contraction table)

APPENDIX B Input Classes

No.	Characteristic of
1	contractions always used in grade 2
2	digits
3	most punctuation and control ($) symbols
4	contractions used after the start of a word
5	$G (grade switch)
6	contractions used at the start of a word
7	isolated full-word contractions
8	$P" (start paragraph in quotation)
9	$P (start paragraph in italics)
10	" (left quote) — outer
11	" (right quote) — inner
12	__ (begin italics)
13	_ (last word of italics)
14	(space), certain control ($) symbols
15	A to J or space occurring in a number
16	contractions always used in grade 2 containing terminal punctuation
17	prefix or first word of compound word
18	non-prefix beginning of word
19	first entry of double entry
20	second entry of double entry
21	"forced" contraction begin sign (/_)
22	units of measure and numbers following numbers
23	Foreign language special symbols
24	Spanish special symbols
25	$FL-SPAN
26	$FL-LIFG
27	decimal point (.) within a number
28	termination sign
29	$G1
30	$G2
31	inner left quote
32	isolated-letter sequences starting with "
33	final quote of an isolated-letter sequence
34	isolated-letter sequence within quotes

APPENDIX C State Variables

No.	Definition (meaning if "true")
1	after the start of a number
2	after the start of a word
3	grade 1 translation
4	in a quotation started by double quote
5	in italicized text
6	not at the start of a prefix or stem
7	part way through a word or phrase too long for one entry
8	just after a space (or A-J), following a number
9	Foreign language passages are Spanish
10	within ~~an inner~~ a quote started by single quote
11	within an isolated-letter group preceded by "
12	not OK to contract lower short words die/ not (after a ~~blank~~ space & one or more =/s or _/s or contractions)

APPENDIX D Right-Context Codes

Code	Class
A	alphabetic characters (letters)
I	letters isolated (not words) when alone
J	punctuation that, if following an isolated letter, implies that a letter sign is needed
L	letters and special symbols preceding or acting as letters
N	numbers
P	punctuation
Q	the double quote character

REFERENCES

Sullivan 75 J. E. Sullivan (ed.), DOTSYS III: A Portable Program for Braille Translation, MTR-2119 (Rev. 1), The MITRE Corporation (October 1975)

Sullivan 76 J. E. Sullivan, The Duxbury Braille Translator User's Manual, Duxbury Systems, Inc., document # DS-JES-760716 (July 1976)