To: Distribution

From: Richard Kissel
Bill Silver

Date: June 17, 1977

Subject: Speedtype

"If y cnn rd tes sb+, te y _mt rd ts doc."

INTRODUCTION

This memorandum describes proposed new commands for word processing. These commands, collectively called Speedtype, allow a user to input text in a shorthand or abbreviated form and then have that text expanded. It is intended that Speedtype become an integral part of Multics WORDPRO.

A prototype of Speedtype exists today on Multics. Users of Speedtype find it allows them to input text more quickly and more accurately. Speedtype has helped them reduce keystrokes by 30% to 50%, thus realizing a significant increase in typing speed.

This memorandum contains the following sections:

- Speedtyping
- Speedtype Features
- Speedtype Enhancements
- Summary of Speedtype Commands
- MPM Documentation
- Appendix A: Speedtype User's Guide

Please send all comments and suggestions on Speedtype to:

Richard J. C. Kissel
Honeywell Information Systems
575 Technology Square.
Cambridge, Mass. 02139

or send Multics mail at M.I.T or System M to:

Kissel.Multics

or call: (617) 492-9319
HVN 261-9319

SPEEDTYPING

Speedtyping, quite simply, is the ability to type a little and get a lot. Typing speed is increased since less is typed.

The primary goal of Speedtype is to allow users to type input data more quickly. Much effort has gone into making Speedtype easy to use and easy to learn. However, each Speedtype design decision was resolved by answering the question, "Which design strategy allows a user to type the fewest keystrokes?" In every case, the design strategy that minimized keystrokes was selected.

Speedtype can also help improve typing accuracy. Typing accuracy is improved by defining and using symbols for words or phrases that are often mistyped. For example, the common typo for "the", "teh", can be corrected automatically by having Speedtype expand the symbol "teh" into "the". Even better, this typo can be eliminated entirely by typing the symbol "t" and having Speedtype expand it into "the".

Speedtype is quite similar to the Multics "abbrev" subsystem which expands command line input. In fact, the motivation for developing Speedtype was to provide an abbreviation/expansion facility that could be used in a word processing environment. The problems Speedtype must solve are how to define, maintain, and list a set of abbreviations that can be typed as input and then expanded.

In order to avoid confusion and ambiguity in terminology between Speedtype and abbrev, the term "abbreviation" is not used when discussing Speedtype. Instead, the term "symbol" is used. All Speedtype commands are named to conform to this terminology.

SPEEDTYPE FEATURES

The primary job of Speedtype is to expand text. This section describes the features of Speedtype that are involved in the expansion process.

Text Segments

Speedtype deals with two types of files, text segments and symbol dictionaries. A text segment contains the input text processed by Speedtype. This processing involves searching through the text segment and expanding all defined symbols. The expanded text is copied into an output text segment.

Speedtype processes an input text segment as just one long character string. The resulting output text segment may also be thought of as one character string. (1) The input string is divided into pairs of tokens. Speedtype recognizes two types of tokens: delimiter tokens and text tokens. Certain ASCII characters are designated as delimiter characters (in general, white space and punctuation characters other than period). All other characters are considered text characters. Figure 1 shows how Speedtype divides an input string into pairs of tokens. Not shown are the special cases that may exist at the beginning and end of an input string where one of the tokens in a pair may be missing.

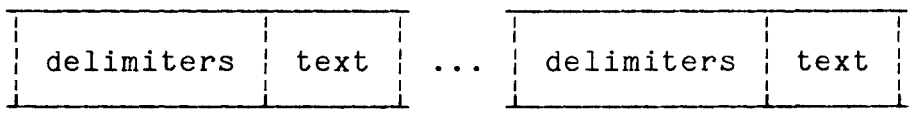


Figure 1. Input String as Pairs of Tokens

The concept of dividing an input string into pairs of tokens (delimiter tokens and text tokens) is a key concept in the general algorithm used to expand an input string. A summary of this algorithm is given below:

1. Get the next delimiter token from the input string.

2. Copy the delimiter token into the output string.

3. Get the next text token from the input string.

4. See if the text token is defined in some table of symbols (in this case a Speedtype symbol dictionary).

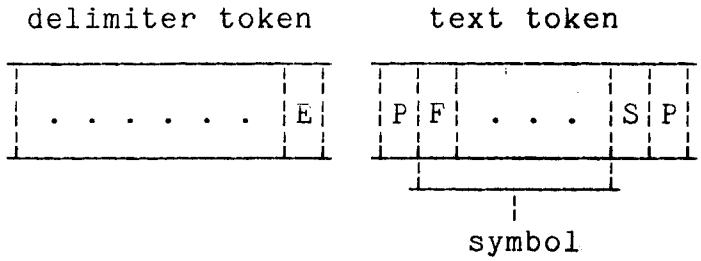
4a. If the text token is not a defined symbol, then copy the text token into the output string.

4b. If the text token is a defined symbol, then get the expansion string that represents this symbol. Copy the expansion string into the output string. The original input symbol is not copied.

5. Repeat the above steps until the end of the input string is reached.

(1) A subroutine interface for string expansion is provided by Speedtype. See the MPM documentation for speedtype_expand_.

Speedtype uses this general expansion algorithm. In addition, special delimiter and text characters, if found in certain positions, result in special processing. Figure 2 shows the detailed format of a pair of tokens.



E => Escape Character
P => Prefix Character(s)
F => First Character
S => Suffix Character
P => Period

Figure 2. Format of a Token Pair

Figure 2 shows that Speedtype performs special processing on the last character of a delimiter token and on the first and last characters of a text token. This special processing is outlined below and discussed in detail later in this section.

Escapes: Certain delimiter characters are recognized as escape characters. If the last character of a delimiter token is an escape character, then special processing is performed on the following text token.

Prefixes: Certain text characters are recognized as prefix characters. If a prefix character is found at the beginning of a text token, then special processing is performed. Recognized prefix characters are not considered part of the symbol. Prefix characters found within the text token cause no special processing and are considered part of the symbol. More than one prefix character may precede the symbol.

Capitalization: If the first character of the symbol is an uppercase letter, then the first letter of the expansion string representing this symbol is capitalized when copied into the output string.

Suffixes: Certain text characters are recognized as suffix characters. If the last character of a text token (after any trailing period is removed) is a suffix character, then special processing is performed. A recognized suffix character is not considered part of the symbol. Suffix characters found within the text token cause no special processing and are considered

part of the symbol. Only one suffix character may follow the symbol.

Period: If the last character of a text token is a period ".", then it is stripped from the text token. The period is copied into the output string after the text token is processed.

Symbol Dictionaries

A symbol dictionary contains all of the information needed by Speedtype to expand an input string. A symbol dictionary is similar to an abbrev ".profile" segment. A symbol dictionary is identified by the entryname suffix ".symbols". Speedtype allows a user to specify the symbol dictionary used. As a default, Speedtype uses a symbol dictionary in the user's home directory. The default symbol dictionary has the pathname:

>udd>Project>Person_id>Person_id.symbols

A symbol dictionary contains three types of information. Speedtype commands allow a user to set, change, and list all of this information. The three types of information are:

Options: Several types of control information are kept in a symbol dictionary. These Speedtype "options" may be set by a user. (See the option_symbols command documentation for a description of the Speedtype options.) The Speedtype options are summarized below:

- Delimiters (except escapes and white space)
- Escape Characters
- Prefix Characters
- Suffix Characters

Symbols: A symbol is a character string that represents a word or phrase. A symbol must be unique within a symbol dictionary. Since symbols are found within text tokens, they may not contain any delimiter characters. The first character of a symbol may not be a prefix character, and the last character of a symbol may not be a suffix character or a period.

Expansions: Every defined symbol has a corresponding expansion string. Expansions do not have to be unique within a symbol dictionary. An expansion may contain any character, including delimiter characters. All suffixing, capitalization, and underlining is performed on expansions, not on symbols. Associated with each expansion is information that specifies how Speedtype is to perform suffixing on that expansion.

The Expansion Process

Speedtype uses the general expansion algorithm described above. However, Speedtype also performs special processing. A more detailed description of how Speedtype expands a token pair is given below:

Delimiters: Processing of the delimiter token involves just copying it into the output string.

Escape Processing: If the last character of the delimiter token is an escape character, then special processing is performed on the following text token. Escape characters contained within the delimiter token are not recognized as escapes. The most important type of escape processing involves inhibiting any processing of the following text token. Instead, the text token is just copied into the output string.

Finding the Symbol: If no escape inhibits the processing of the text token, then the next step is to find the symbol contained in the text token. This involves stripping off any prefix characters, suffix character, or trailing period. If no symbol is found within the text token, i.e., it consists of just prefix and/or suffix characters, then no further processing is performed on this text token and it is copied as is into the output string.

Decapitalization: If the text token contains a symbol, then Speedtype decapitalizes it. This involves testing the first character of the symbol, and if it is an uppercase letter, translating it to lowercase. This translation is actually performed on a temporary copy of the symbol. The original input symbol is not modified.

Expansion: Speedtype then takes the decapitalized symbol and searches for it in the current symbol dictionary. If found, the expansion for this symbol is copied into the output string, otherwise the original input symbol (and any suffix character) is copied.

Capitalization: If the input symbol was decapitalized and replaced by its expansion, then Speedtype capitalizes the expansion string copied into the output string. This involves testing the first character of the expansion string, and if it is a lowercase letter, translating it to uppercase.

Suffix Processing: If a suffix character was stripped from the symbol, and if the symbol was expanded, then Speedtype performs suffixing on the expansion string copied into the output string. This processing depends upon the suffix and how the suffix is defined for this symbol.

Prefix Processing: If any prefix characters were stripped from the symbol, then Speedtype performs prefix processing on the symbol or expansion string copied into the output string. Prefix processing is always performed after any capitalization or suffixing.

Period Processing: If a period was stripped from the symbol, then it is added to the output string after all other processing of the text token is performed.

Escapes

The escapes recognized by Speedtype are listed below. The actual escape characters recognized are defined in a symbol dictionary and may be set by the user. Listed with each escape is its name and its default character. The special processing performed for each escape is also described.

temp (~) The temp (temporary) escape is the standard Speedtype escape. It causes Speedtype to not process the following text token. Thus this escape can be used to prevent a symbol from being expanded and can prohibit prefix processing for the next text token. Instead, the text token is copied as is into the output string. The temp escape character itself is not copied into the output string.

pad (octal 177) The pad escape is useful in situations where an input text segment is also used as the output text segment and is expanded over and over. The effect of this escape is the same as that for the temp escape. However, unlike the temp escape, this escape character is copied into the output string. The default character used for the pad escape is the ASCII PAD character (octal 177). Even though this character is copied into the output string, it will not show up when printed. Users are cautioned that the presence of a PAD character in the text segment may cause problems during subsequent editing.

perm (`) The perm (permanent) escape is a convenient way for a user to enter a pad escape. The effect of this escape is the same as the temp escape, and like the pad escape, it is copied into the output string. However, the perm escape character is then converted to the pad escape character.

trans (:) The function of the trans (transparent) escape is to concatenate text tokens that are processed separately. The trans escape character is not copied into the output string. The following text token is processed as if no escape was recognized. Any prefix processing performed on the previous text token is continued and performed on the next text token. Additional prefix processing may be specified.

space (;) The function of the space escape is to generate spaces (ASCII blanks) in the output string. The processing of this escape is conditional on the first characters of the following text token. If the following text token begins with one or two numeric characters (numbers from 0 to 99), then the space escape character and these numeric characters are replaced in the output string with the specified number of spaces. For example, ";5" is replaced by five spaces in the output string. The rest of the text token is then processed normally. If the following text token does not contain a number as specified above, then the space escape character remains unchanged in the output string and the following text token is processed as if no escape was recognized.

Suffixes

Suffix processing is performed only on defined symbols. If a symbol is not defined, or if the specified suffix is turned off for the symbol, then no suffix processing is performed. Instead, the symbol and the suffix character are copied as is into the output string.

Appending a suffix to a symbol's expansion string is done in several different ways depending upon how the suffix is defined for the symbol. The normal way is to just add the suffix string associated with the suffix directly to the expansion string. However, to accommodate the many anomalies of the English language, such tricks as dropping the last letter, doubling the last letter, adding letters, etc., may be performed on the expansion string in order to add a suffix string.

A user has considerable control over how Speedtype performs suffixing. (See the add_symbols documentation for a description of how Speedtype performs suffixing.) A user may disable suffixing for a given symbol, or just disable one or more suffixes for that symbol. A user may also specify a different way to process a suffix for a symbol.

The suffixes currently recognized by Speedtype are listed below. The actual characters representing the suffixes are defined in a symbol dictionary and may be set by the user. Except for "plural", the suffix string associated with each suffix is the suffix itself. Also listed below with each suffix is the default character used to represent that suffix.

SUFFIX	STRING	
(plural)	"s"	(+)
ed	"ed"	(-)
ing	"ing"	(*)
er	"er"	(=)
ly	"ly"	(¦)

Prefixes

Prefix processing is performed on the text token string copied into the output string. It is performed regardless of whether symbol expansion was performed, and is always performed after capitalization and suffixing have been performed.

The prefixes recognized by Speedtype are listed below. The actual prefix characters recognized are defined in a symbol dictionary and may be set by the user. Listed for each prefix is its name and its default character. The special processing performed for each prefix is also described.

under (_) The function of the under (underline) prefix is to underline the output string. The underlining is performed by taking each character of the output string and adding, in a canonical way, a backspace character and an underscore character. The resulting underlined string is in canonical form. Underlining is not performed if the output string already contains backspace characters.

upper (+) The function of the upper (uppercase) prefix is to translate the output string into uppercase. Each lowercase letter in the output string is translated to uppercase. Characters that are not lowercase letters are not changed. If both the upper and under prefixes are recognized, then regardless of the order in which they are specified, uppercase processing is performed first.

SPEEDTYPE ENHANCEMENTS

Speedtype, in its present form, is a useful word processing tool. However, it has several limitations that in the future should be eliminated. The most important of these current limitations are:

- the inability to allow concurrent users to safely update and use a symbol dictionary.
- a maximum expansion length of 56 characters.
- a maximum number of symbols, currently about 1000.
- the processing of only 5 suffixes.
- the inability to specify nonstandard suffix expansions, for example, the plural of child.
- the requirement that expansion involve a whole input segment.

Many of these current Speedtype limitations could be eliminated by reimplementing Speedtype symbol dictionaries as indexed files (accessed via vfile_). This would allow a shared symbol dictionary to be updated by one user while at the same time being used for expansion by other users. This would also allow for longer expansions and a virtually unlimited number of symbols. The suffixing algorithm could be improved to allow for more kinds and more complex suffixes.

One enhancement to Speedtype that is already planned is to allow Speedtype expansion while editing. The new WORDPRO text editor (see MTB-339) allows for Speedtype expansion in regular expressions, replacement strings, and in input text. This method of using Speedtype makes it unnecessary to expand a whole text segment in order to process just a few symbols. Thus text segments will not be expanded over and over and the pad escape will not be needed.

Although Speedtype was designed for use in a word processing environment where the language involved is English, Speedtype could be used to input and update other natural languages (French, Spanish, etc.) and common computer languages (PL/I, COBOL, FORTRAN, etc.). Speedtype has already been used with great success in the processing of Multics "exec_com" programs. A Speedtype command could be implemented that would expand PL/I source. It would process PL/I statements using a PL/I symbol dictionary and PL/I comments using an English symbol dictionary.

SUMMARY OF SPEEDTYPE COMMANDS

This section presents a summary of the Speedtype commands. They are grouped according to function.

Symbol Dictionary Maintenance

add_symbols	adds symbols to the current symbol dictionary.
change_symbols	changes the expansion or suffixing of a symbol in the current symbol dictionary.
delete_symbols	deletes symbols from the current symbol dictionary.
find_symbols	finds and lists symbols in the current symbol dictionary that represent specified expansions.
list_symbols	lists symbols in the current symbol dictionary.
option_symbols	sets Speedtype options in the current symbol dictionary.

Symbol Dictionary Selection

print_symbols_path	prints the pathname of the current symbol dictionary.
use_symbols	sets the current symbol dictionary.

Symbol Expansion

expand_symbols	expands all the symbols in a specified text segment.
retain_symbols	retains all symbols in a specified text segment by placing a Speedtype escape in front of each symbol.
show_symbols	expands an input string and prints the output string.

MPM DOCUMENTATION

The remainder of this memorandum presents draft MPM documentation for the Speedtype commands.

Name: add_symbols, asb

The add_symbols command adds a symbol to the current symbol dictionary. All suffixes are enabled for the added symbol.

Usage

add_symbols symbol expansion {-control_args}

where:

1. symbol
   is the symbol to be added. Its length must be 7 characters or less and it may not contain delimiter characters. Its first character may not be a defined prefix character or a capital letter, and its last character may not be a defined suffix character or a period.

2. expansion
   is the expansion string that replaces the symbol. The length of the expansion string must not exceed 56 characters. The expansion string may contain any characters. If the expansion string contains spaces and/or tabs, then it must be enclosed in quotes.

3. control_args
   may be chosen from the following:

   -force, -fc
   specifies that the replacement of an existing symbol should be done without question. If the symbol is already defined, and this argument is not specified, then the user is asked to authorize the replacement of the symbol.

   -suffix STR
   enables or disables suffixing for this symbol. STR must be either "on" or "off". If STR is "on", then suffixing is enabled and all suffixes are processed according to the default rules described in the notes below. If STR are "off", then all suffixes are disabled for the symbol. If this control argument is not specified, then "on" is assumed.

-plural STR
defines the plural suffix for this symbol. STR must be "on" or "off" or a string that can be used as the plural of the expansion of this symbol. If STR is the plural of this symbol, then it must be an expansion string that can be generated by Speedtype when using any of its known rules for processing the plural suffix. Otherwise, the plural suffix is disabled for this symbol and a warning message is printed. If STR is "on", then the plural suffix is enabled for this symbol and processed according to the default rules for the plural suffix. If STR is "off", the plural suffix is disabled for this symbol.

-ed STR
defines the "ed" suffix for this symbol. This control argument follows the same rules as the -plural control argument.

-ing STR
defines the "ing" suffix for this symbol. This control argument follows the same rules as the -plural control argument.

-er STR
defines the "er" suffix for this symbol. This control argument follows the same rules as the -plural control argument.

-ly STR
defines the "ly" suffix for this symbol. This control argument follows the same rules as the -plural control argument.

Notes

The default rule for appending a suffix string to an expansion string is a function of the suffix and the word type of the expansion string.

The word type of the expansion string is determined from its last characters. The characters "C" and "V" are used below to represent consonants and vowels. The character "X" is used to represent any character. The word types recognized and the suffix strings used are listed below:

add_symbols

WORD TYPES

0 - other (=> none of the below)
1 - "XCe"
2 - "XVe"
3 - "XCy"
4 - "XVy"
5 - "Xch", "Xsh", or "Xex"
6 - "CVC"

SUFFIX STRINGS

1 - "s" (plural)
2 - "ed"
3 - "ing"
4 - "er"
5 - "ly"

The actions performed by Speedtype when adding a suffix string to an expansion string are listed below:

ACTIONS

1 - Add suffix string directly
2 - Drop last character, add suffix string
3 - Double last character, add suffix string
4 - Replace last character with "i", add suffix string
5 - Replace last character with "ie", add suffix string
6 - Add "e", add suffix string

The suffix action table presented below shows the action performed by Speedtype when adding a specified suffix string to an expansion string of a given word type.

SUFFIX ACTION TABLE

WORD TYPE	SUFFIX 1	2	3	4	5
0	1	1	1	1	1
1	1	2	2	2	2
2	1	2	1	2	1
3	5	4	1	4	1
4	1	1	1	1	1
5	6	1	1	1	1
6	1	3	3	3	1

Name: change_symbols, csb

The change_symbols command changes the expansion or suffixing of a specified symbol. Control arguments are processed one at a time. Specifying more than one control argument has the same effect as issuing the command several times with one control argument each time.

Usage

change_symbols symbol {-control_args}

where:

1. symbol
   is the symbol changed. This symbol must be defined in the current symbol dictionary.

2. control_args
   may be chosen from the following:

   -exp STR
   where STR represents the new expansion string for this symbol. This control argument does not change the way suffixing is performed for the symbol.

   -suffix STR
   -plural STR
   -ed STR
   -ing STR
   -er STR
   -ly STR
   the above control arguments work the same way as described for the add_symbols command.

Name: delete_symbols, dsb

The delete_symbols command deletes the specified symbols from the current symbol dictionary.

Usage

delete_symbols symbols

where symbols are the symbols deleted from the current symbol dictionary.

Name: expand_symbols, esb

The expand_symbols command takes an input text segment and expands it using the options and symbols defined in the current symbol dictionary.

Usage

expand_symbols input_path {output_path}

where:

1. input_path
   is the pathname of the input text segment.

2. output_path
   is an optional pathname of an output text segment. If no output pathname is specified, then the text segment specified by input_path is used as the output text segment. The original contents of the input text segment are overwritten.

Name: find_symbols, fsb

The find_symbols command finds and then lists all of the symbols associated with specified expansions contained in the current symbol dictionary. One or several or all expansions may be listed.

Usage

find_symbols {expansions} {-control_args}

where:

1. expansions
   are optional arguments that specify expansions to find and list. If an expansion is represented by more than one symbol, then all of its symbols are found and listed. If any given expansion is not found, then a message is printed stating that the expansion is not defined. If no expansions are specified, then all expansions in the current symbol dictionary are listed. The expansions are listed in order according to ASCII collating sequence.

2. control_args
   can be chosen from the following:

   -long, -lg
   specifies that for each symbol listed, its expansion string with suffixing is listed for each suffix enabled for that symbol.

   -option, -op
   specifies that all option information for the current symbol dictionary is to be listed. If this is the only control argument specified, then only the option information is listed.

   -total, -tt
   specifies that the total number of symbols defined in the current symbol dictionary is to be printed. If this is the only control argument specified, then only the total is printed.

Name: list_symbols, lsb

The list_symbols command lists one or several or all of the symbols defined in the current symbol dictionary.

Usage

list_symbols {symbols} {-control_args}

where:

1. symbols
   are optional arguments that specify the symbols to list. If any given symbol is not found, then a message is printed stating that the symbol is not defined. If no symbols are specified, then all symbols in the current symbol dictionary are listed. They are listed in order according to ASCII collating sequence.

2. control_args
   can be chosen from the following:

   -long, -lg
   specifies that for each symbol listed, its expansion string with suffixing is listed for each suffix enabled for that symbol.

   -option, -op
   specifies that all option information for the current symbol dictionary is to be listed. If this is the only control argument specified, then only the option information is listed.

   -total, -tt
   specifies that the total number of symbols defined in the current symbol dictionary is to be printed. If this is the only control argument specified, then only the total is printed.

Name: option_symbols, osb

The option_symbols command allows a user to change certain optional control information in the current symbol dictionary. This information is summarized in the notes below.

Usage

option_symbols {-control_args}

where:

1. control_args
   can be chosen from the following:

   -delim STR
   specifies a new set of delimiter characters. None of the characters in this string may be currently defined escape, prefix, or suffix characters.

   -pad X
   -perm X
   -temp X
   -trans X
   -space X

   -under X
   -upper X

   -plural X
   -ed X
   -ing X
   -er X
   -ly X
   the above control arguments set the corresponding escape, prefix, or suffix characters recognized by Speedtype.

Notes

Presented below is a summary of all Speedtype options. The default character(s) used to represent each option is shown on the right.

```
Delimiters:
     Escapes (see below)
     White Space  (Break, Tab, New Line)
     Others  ","()?!<>[]{}"

Escapes:
     pad       (octal 177)
     perm      "`"
     temp      "~"
     trans     ":"
     space     ";"

Prefixes:
     under     "_"
     upper     "+"

Suffixes:
     plural    "+"
     ed        "-"
     ing       "*"
     er        "="
     ly        "|"
```

Name: print_symbols_path, psbp

The print_symbols_path command prints the pathname of the current symbol dictionary.

Usage

print_symbols_path

Name: retain_symbols, rsb

The retain_symbols command takes an input text segment and inserts Speedtype escape characters wherever symbols would be expanded if this text segment were being processed by the expand_symbols command. All symbols in the text segment are thus retained during future expansion.

Usage

```
retain_symbols input_path {output_path} {-control_args}
```

where:

1. input_path
   is the pathname of the input text segment.

2. output_path
   is the optional pathname of an output text segment. If no output pathname is specified, then the text segment specified by input_path is used as the output text segment. The original contents of the input text segment are overwritten.

3. control_args
   can be chosen from the following:

   -perm
   specifies that the perm escape character is to be used. If no control argument is specified, then -perm is assumed.

   -temp
   specifies that the temp escape character is to be used. Specifying this control argument causes the symbols in the output text segment to be retained for only one expansion.

Notes

In addition to inserting the specified escape character wherever necessary, all existing "Pad" escapes are converted to the specified escape. This allows for more convenient editing of the input text segment, since all escape characters are thus printable.

Name: show_symbols, ssb

The show_symbols command shows how Speedtype expands an input string. The expansion is performed using the options and symbols in the current symbol dictionary. The expanded string is printed on the user's terminal.

Usage

show_symbols term1 ... termi

where:

1. term1 ... termi
   are arguments that are concatenated into the input string that is expanded. These terms are separated in the input string by one space. If other spacing is desired, the input string should be enclosed in quotes.

Name: use_symbols, usb

The use_symbols command sets the current symbol dictionary. All Speedtype commands will then use this symbol dictionary. If this symbol dictionary does not exist, then the user is asked if it *should be* created.

Usage

```
use_symbols path
```

where:

1. path
   is the pathname of the symbol dictionary that is to be the new current symbol dictionary. If the entryname suffix ".symbols" is not specified, then it is added.

Notes

If other Speedtype commands are issued in a user's process before the use_symbols command, then those commands use the default symbol dictionary in the user's home directory. The default symbol dictionary has the pathname:

```
>udd>Project>Person_id>Person_id.symbols
```

Name: speedtype_expand_

The speedtype_expand_ subroutine takes an input text string and expands it using the options and symbols defined in the current Speedtype symbol dictionary. It returns the expanded output string.

Usage

```
dcl  speedtype_expand_ entry (ptr, fixed bin(21), ptr,
     fixed bin(21), fixed bin(21), fixed bin(35));

call speedtype_expand_ (in_string_ptr, in_string_len,
     out_buf_ptr, out_buf_len, out_string_len, code);
```

where:

1. in_string_ptr (Input)
   is a pointer to the input string expanded.

2. in_string_len (Input)
   is the length in characters of the input string.

3. out_buf_ptr (Input)
   is a pointer to a buffer where the output string is returned.

4. out_buf_len (Input)
   is the length in characters of the output string buffer.

5. out_string_len (Output)
   is the actual length in characters of the expanded output string.

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

SPEEDTYPE USER'S GUIDE

This user's guide is intended to help WORDPRO users learn to use Speedtype. A narrative description of how and why to use each Speedtype command is presented. (1) Examples of how symbols are expanded are also provided.

Special Instructions

This appendix is included in this memorandum in order to encourage readers to use Speedtype. Since the Speedtype commands are not yet installed Multics commands, you must temporarily set up to use a private version of Speedtype maintained by the authors. Users of the M.I.T. or System M Multics systems will be able to use a prototype version of Speedtype by issuing the following Multics command:

```
! asr >udd>Multics>Kissel>search
```

This command adds a directory to your search rules that contains the Speedtype commands (or links to them). Doing this is better than copying the Speedtype object code since you will be able to take immediate advantage of new versions of Speedtype that contain bug fixes or other improvements.

Also contained in this directory is a Speedtype symbol dictionary that you should copy. It contains symbols for about 70 of the most common English words. After copying this symbol dictionary, you can change or delete the symbols you do not like. You can then begin adding your own symbols. The pathname of this symbol dictionary is:

>udd>Multics>Kissel>search>english.symbols

Defining The Current Symbol Dictionary

All Speedtype commands use your current symbol dictionary. The first time you execute a Speedtype command in your process the default symbol dictionary in your home directory is defined as your current symbol dictionary. If you want to know the pathname of your current symbol dictionary use the command, print_symbols_path:

---

(1) Throughout this user's guide, the exclamation mark (!) is printed at the beginning of every line typed by the user. This is done only to distinguish user entries from system-generated printouts; the user should not actually begin an entry with an exclamation mark.

```
! psbp
  >udd>Multics>Kissel>search>english.symbols
```

If you want a different symbol dictionary to be your current symbol dictionary, then use the command, use_symbols:

```
! usb path
```

where path is the pathname of the new symbol dictionary you want to use. If the symbol dictionary you want to use does not exist, then Speedtype asks you if you want it created.

In order to start using symbols in an input text segment, memorize about 5 symbols. Start with simple and common words such as: "the", "and", "is", "to", and "with". You will find that symbols are like peanuts, you cannot stop with just a few. In a short time you will be using a hundred or more symbols and using prefixing, suffixing, capitalization, and other Speedtype features.

Listing Information In A Symbol Dictionary

Once you have established your current symbol dictionary, you will want to list the symbols that are defined in it. There are two Speedtype commands that list information in a symbol dictionary. One command, list_symbols (lsb), is oriented toward symbols while the other, find_symbols (fsb), is oriented toward expansions.

For example, if you want to know if the term "ex" is a defined symbol, use the list_symbols command as shown below:

```
! lsb ex
  ex        example
```

If you want to know if a symbol is defined for the word "example", use the find_symbols command as shown below:

```
! fsb example
  example  -  ex
```

If either of these commands are issued without the optional argument, then all symbols or all expansions in the current symbol dictionary are listed.

Defining A Symbol

There are Speedtype commands that add, delete, and change symbols. They are called, appropriately enough, add_symbols (asb), delete_symbols (dsb), and change_symbols (csb). For example, if you want to define the symbol "fw" for the word "follow", issue the command:

```
! asb fw follow
```

You can then use the list_symbols command to list this new symbol. This command is shown below with the "-long" control argument, thus all defined suffix expansions for this symbol are listed:

```
! lsb fw -lg
  fw          follow
        (+) follows
        (-) followwed
        (*) followwing
        (=) followwer
        (|) followly
```

As you can see, the spelling of the word "follow" with the suffixes "ed", "ing", "er" is incorrect. The suffix "ly" does not even make sense for the word follow. You can change the incorrect suffixes and delete the useless suffix by issuing the following change_symbols commands:

```
! csb fw -ed followed -er follower
! csb fw -ing following
! csb fw -ly off
```

Now you may list this symbol again. In the example below it is listed with the expansion first.

```
=> fsb follow -lg
   follow - fw
        (+) follows
        (-) followed
        (*) following
        (=) follower
```

If you no longer want the symbol "fw" defined, delete it by issuing the delete_symbols command:

```
! dsb fw
```

Changing Symbol Options

All of the special characters used by Speedtype (see the option_symbols command) can be changed. You cannot change the way these Speedtype features work, but you can change the actual characters recognized by Speedtype.

For example, suppose you were typing on a terminal that did not have the temp escape character "~". You could substitute another character, say "&", and have it become the temp escape character. This can be done with the option_symbols command shown below:

```
! osb -temp &
```

If you want to list the options and special characters defined in the current symbol dictionary, issue the list_symbols or find_symbols command with the "-option" control argument.

Expanding An Input Segment

This section describes the normal method of operation that you should use when preparing documentation with Speedtype.

For this example, assume that you want to produce a document in the form of a runout segment, call it test.runout. Type your input text into the segment, test.runoff. This segment can be expanded by issuing the expand_symbols command shown below:

```
! esb test.runoff
```

After this command is executed, test.runoff contains the expanded text. It may then be used as input to the "runoff" command.

Escaping An Input Segment

One of the important features of Speedtype is that it allows you to specify that a symbol (text token) be retained and not be expanded. This is specified by using one of the Speedtype escapes.

In general, you should not define a symbol that is itself a valid English word. For example, "i" may seem like a good symbol to use for the word "in", however, since "I" is itself a common word, defining "i" as a symbol would lead to confusion.

However, even if you choose your symbols carefully, there are still situations where a symbol must left unexpanded. Such situations might involve the initials of a person's name, examples within the text, or special numbering schemes that use letters. When typing input you must recognize these situations and use the appropriate Speedtype escapes.

Editing Existing Text Segments

One common situation that involves escapes is when you have to edit an existing segment. If this segment was not typed by someone who was aware of the symbols that you have defined, then you must insert the appropriate escapes into the segment before you expand it. This may be done automatically by using the retain_symbols command. This command saves you the trouble of searching through a segment for symbols that would be expanded. It inserts escape characters wherever text would be expanded by the expand_symbols command.

A good policy when beginning to edit an existing segment that may contain some of your symbols is to insert escapes in front of those symbols by issuing the retain_symbols command as shown below:

```
! rsb existing.runoff
```

Examples Of Expansion

The Speedtype command, show_symbols (ssb), can be used to show how Speedtype expands an input string. This command is similar to the expand_symbols command except that it expands an input string rather than an input text segment. The expanded output string is printed on your terminal.

The show_symbols command is especially useful when first learning how to use Speedtype. It will enable you to practice using Speedtype without having to use a text editor. The show_symbols command is used below in examples that show how Speedtype performs expansions.

The following examples define a symbol and then show how this symbol is expanded with suffixing and capitalization.

```
! asb c call
! ssb c
  call

! ssb c+ c- c* c= c¦
  calls called calling caller callly

! ssb C C*
  Call Calling
```

The following examples show how you can turn off certain suffixes for a particular symbol. It shows what Speedtype does with symbols that are not defined. It also shows what Speedtype does with tokens that do not contain a symbol.

```
! csb c -ly off
! ssb c c¦
  call c¦

! ssb undefined_symbol
  undefined_symbol

! ssb _* .
  _* .
```

The following examples show how the suffix "ly" may be used.

```
! asb qt quiet -suffix off -ly on
! ssb qt qt¦
  quiet quietly
```

The following examples show the various ways that suffixes can be appended to an expansion. Several more symbols are defined. The expansions of these symbols represent various word types.

```
! asb sp space
! asb ht hit
! asb cp copy
! asb sx suffix
```

```
! ssb sp sp*
  space spacing

! ssb ht ht=
  hit hitter

! ssb cp cp- cp+
  copy copied copies

! ssb sx sx+
  suffix suffixes
```

The following examples show how the temp, perm, and special escapes are used. Notice that escapes only work when they are the last character of a delimiter token. These examples use the symbols defined above.

```
! ssb c ~c ~~c
  call c ~c

! ssb ht=~'s
  hitter's

! ssb `c `~`c
  c `~c
```

The following examples show how the trans escape can be used to combine terms that are expanded separately.

```
! asb w with
! asb ot out

! ssb w:ot
  without

! ssb sp:person
  spaceperson
```

The following examples show how the space escape can be used to insert spaces (blanks) into your text. It also shows that no more than 99 spaces can be inserted.

```
! ssb "a;5b;11c"
  a     b           call

! ssb "a; b;100c"
  a; b;100c
```

The following examples show how you can use Speedtype to underline words or phrases.

```
! asb iot "in order to"

! ssb _Iot
  In order to

! ssb "_C* _w:ot"
  Calling without
```

The following examples show how you can use Speedtype to translate a word or phrase to uppercase.

```
! ssb +iot
  IN ORDER TO

! ssb +C*
  CALLING

! ssb +test_5
  TEST_5
```

The following examples show how Speedtype can be used to perform both types of prefix processing.

```
! ssb _+w:ot
  WITHOUT

! ssb +_C*
  CALLING

! ssb +_undefined
  UNDEFINED
```

```
! usb example.symbols

! lsb
Total:  17 symbols
al          all
e           the
fl          final
fm          from
ft          first
fw          follow
gt          great
lt          last
oy          only
ph          path
s           is
stnd        stand
t           to
ta          than
un          under
wl          will
y           you

! print poem.example

        poem.example

   If y fw al e ph+,
   Fm e ft t e lt,
   Y wl fl¦ un:stnd,
   Oy e _+root s gt= ta.

! esb poem.example

! file_output reader.mind; print poem.example
```