ffortid Program Ver 4.0 Decomposition Manual

Harry I. Hornreich

Technion - Israel Institute of Technology

ffortid Program Ver 4.0 Decomposition Manual

Harry I. Hornreich

Technion - Israel Institute of Technology

ABSTRACT

This manual describes the decomposition of the ffortid dtroff postprocessor program into Software Units (SWU). Each SWU has a page describing its name, number, type, source code, scope diagram, capabilities, interface and service flow diagram (SFD). A SWU is any functional piece of software code. It can provide different kinds of services to other SWU depending on the software language semantics it is written in. Example services (in C) are declarations, definitions, global variables, functions & procedures. The SWU page captures its origin (its source code), its scope (the SWU it is composed of), its capabilities i.e. the services it offers to other SWU, its interface i.e. how its services can be accessed and how they can, might or should affect the enviroment in which they are used. The SFD captures graphically the relationships between the sub-units the SWU is composed of and its enviroment. The SFD shows not only the flow of data but also the use of declarations, definitions, procedure calls and any other kind of software service.

1. Overview

1.1. ffortid History

The first author of ffortid was Cary Buchman, an M.Sc. student at UCLA, and the first version was written during the years 1983-1984. That version could handle only Hebrew though it did have some hooks for Arabic that proved to be useless. The first external customer was the Hebrew University. Mulli Bahr a guru from HU modified the code to optimize the output in 1986 during a visit to UCLA. Johny Srouji extended ffortid for Arabic in 1989-1991.

Version	Years	Author	From	Major Modification
1.0	1983-1984	Cary Buchman	UCLA	Hebrew
2.0	1986	Mulli Bahr	HU	Output Optimization
3.0	1989-1991	Johny Srouji	Technion	Arabic
4.0	1995	Harry Hornreich	Technion	Letter Stretching

An up to date manual page of ffortid can be found at the end of this manual.

1.2. ffortid File Statistics

Num	File	Length (lines)	Functions
1	lex.h	31	-
2	lex.dit	38	-
3	token.h	39	-
4	macros.h	33	-
5	table.h	18	-
6	dump.c	917	15
7	lines.c	372	10
8	main.c	631	1
9	misc.c	114	4
10	width.c	596	14
Total		2789	44

2. ffortid Program Software Units Summary

Num	Name	Type	Size (lines)	Low-Level
1	ffortid	Program	3803	
2	Dump	Module	1020	
3	Lines	Module	493	
4	Main	Module	1457	
5	Misc	Module	204	
6	Width	Module	629	
7	token.h	Declarations source file	39	*
8	lex.h	Definitions source file	31	*
9	macros.h	Definitions source file	33	*
11	dump.c	Source file	917	
12	table.h	Declarations source file	18	*
13	lines.c	Source file	372	
14	lexer	Lex generated source file	705	
15	main.c	Source file	631	
16	misc.c	Source file	114	
17	width.c	Source file	596	
18	dump_defin	Definitions block	30	*
19	dump_line	Procedure	125	*
20	reverse_line	Procedure	87	*
22	print_line	Procedure	21	*

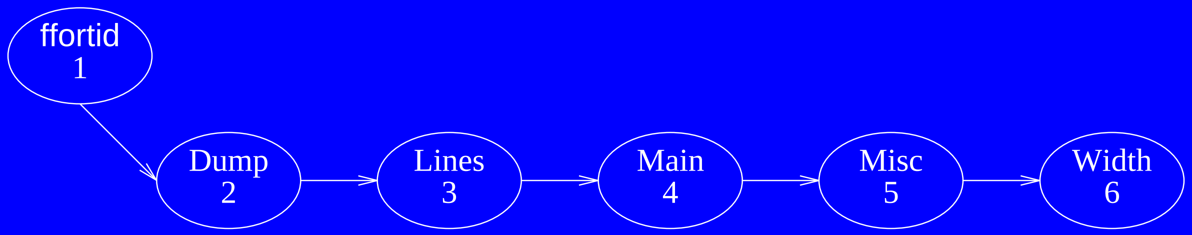
Table continued on next page ...

Num	Name	Type	Size (lines)	Low-Level
23	lines_defin	Definitions block	33	*
24	new_free_token	Function group	91	*
25	insert_tokens	Procedure group	74	*
26	put_tokens	Procedure group	135	*
27	lexer.dit	Lex source file	38	*
28	main_defin	Definitions block	73	*
29	main	Function	558	*
30	new_font	Procedure	42	*
31	font_info	Procedure	42	*
32	out_of_memory	Procedure	17	*
33	yywrap	Function	13	*
35	width_defin	Definitions block	51	*
36	init_dev_font	Procedure group	260	*
37	width_calc	Function group	151	*
38	debug_error	Procedure group	82	*
39	recalc_horiz	Procedure	47	*
40	calc_total	Function	55	*
42	inquire_token	Function group	45	*
43	stretch	Function group	607	
44	stretch_a_line	Function group	182	*
45	stretch_candidates	Function group	131	*
46	stretch_a_word	Function group	116	*
47	spread_stretch	Function group	123	*
48	char_info	Function group	52	*

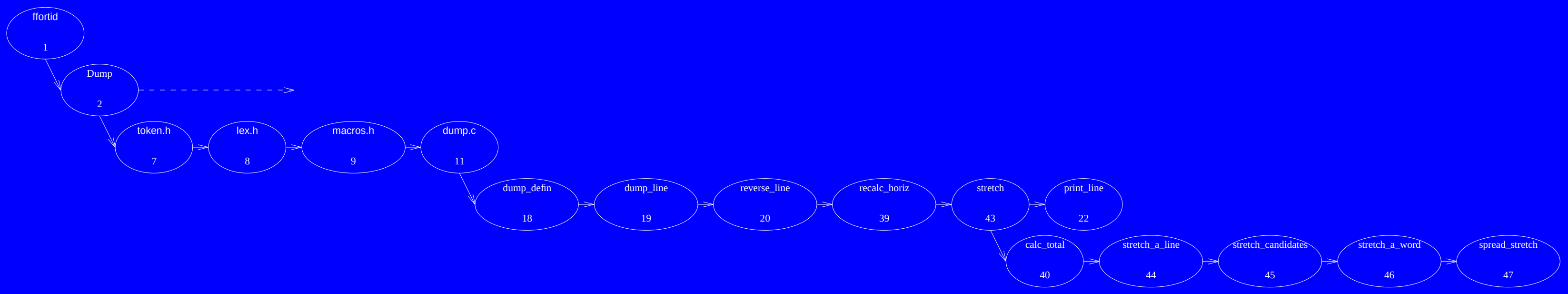
3. Software Units Overview and Pages

In the following pages is an overview of all the SWU in the decomposition followed by a SWU page describing each one in depth. The numbers used in the overview are the SWU numbers given in the previous page. The notation followed in the SFD in the SWU pages is described in Appendix A.

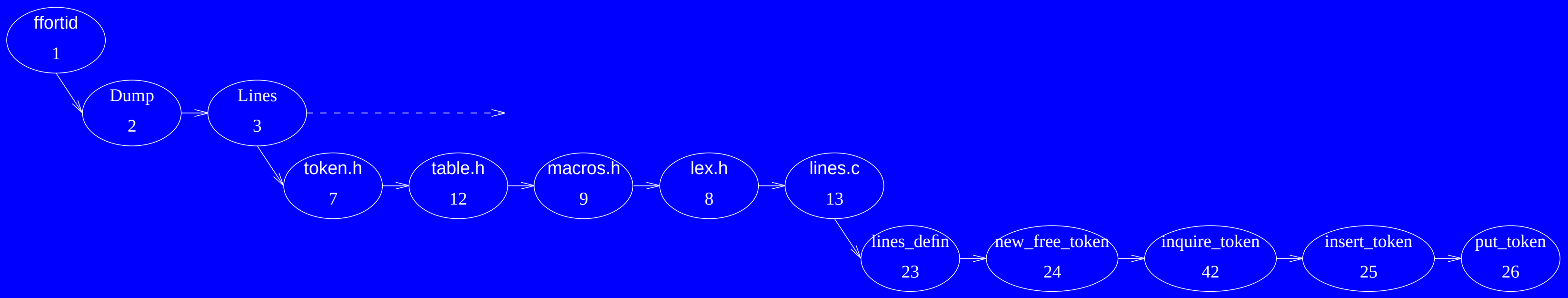
ffortid Software Units Decomposition - Top Level



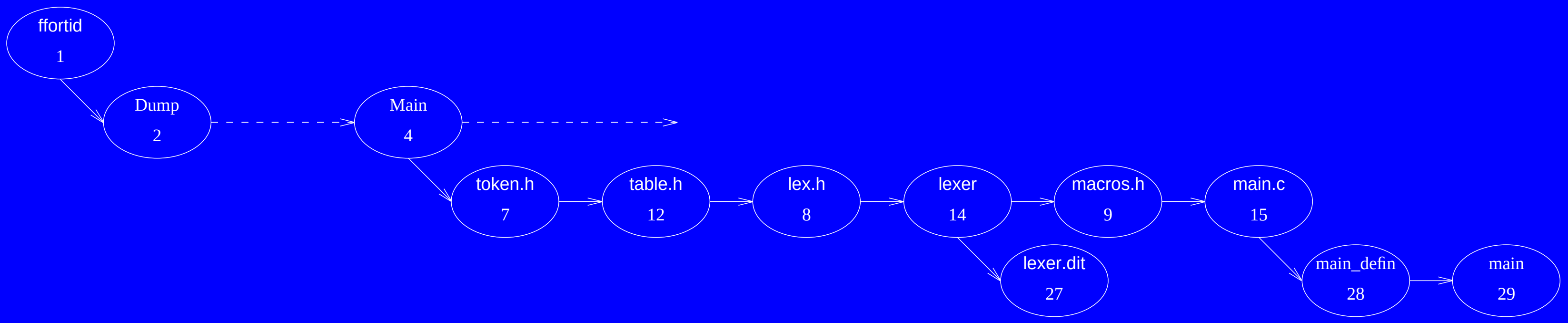
ffortid Software Units Decomposition - Cont 1/5



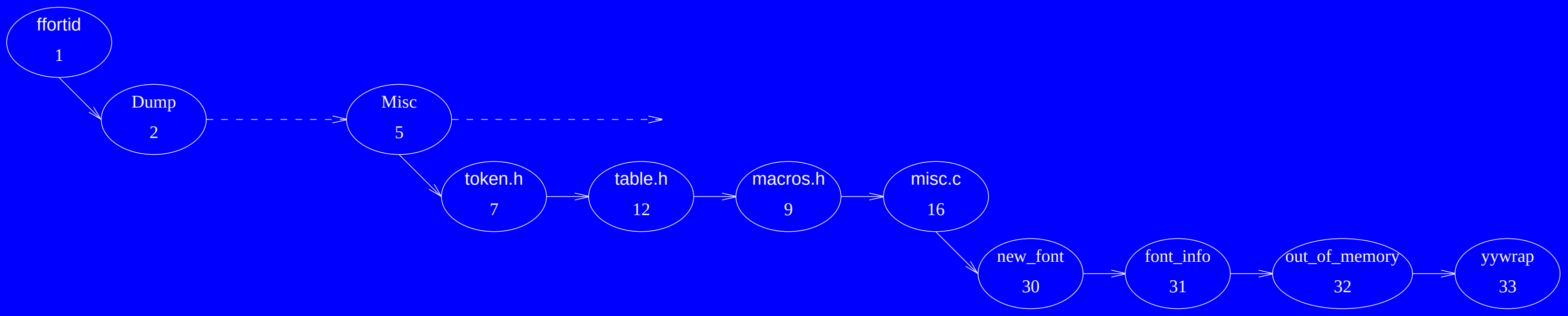
ffortid Software Units Decomposition - Cont 2/5



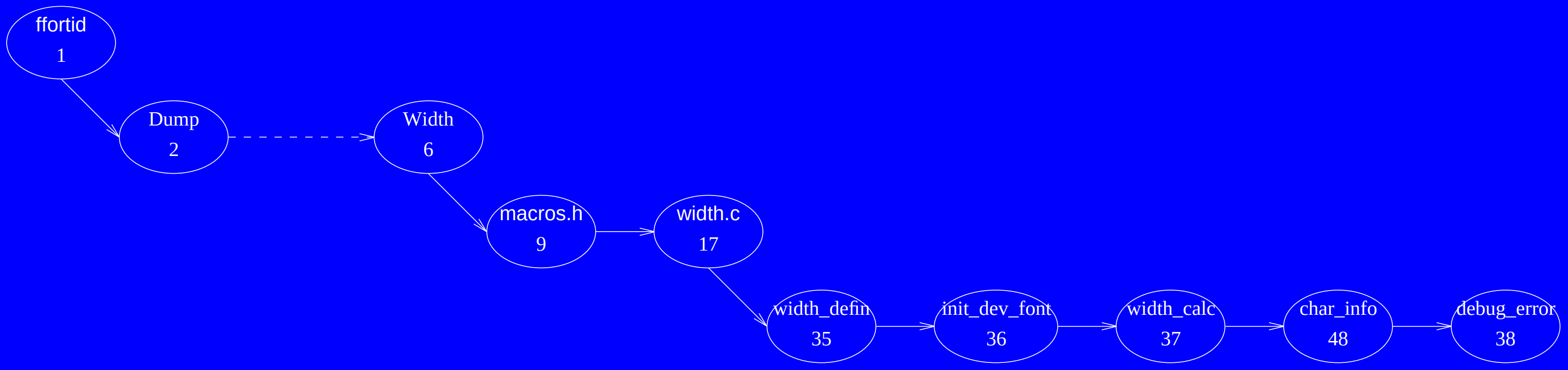
ffortid Software Units Decomposition - Cont 3/5



ffortid Software Units Decomposition - Cont 4/5



ffortid Software Units Decomposition - Cont 5/5

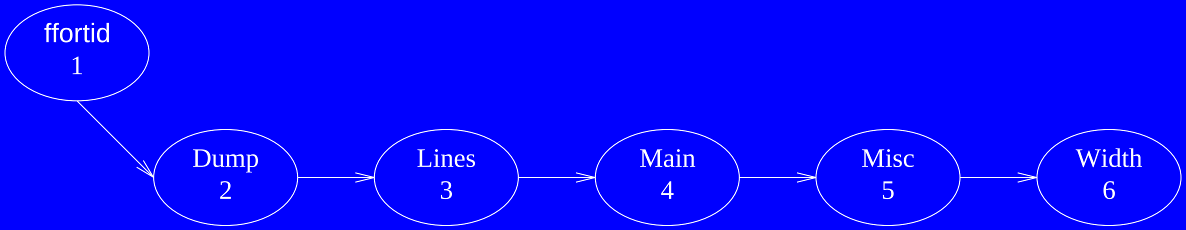


Software Unit #1 — ffortid

1.1 Software Unit Type

Program. (lex.h, lex.dit, token.h, macros.h, table.h, dump.c, lines.c, main.c, misc.c, width.c)

1.2 Scope Diagram



1.3 Capabilities

ffortid takes from its standard input dtroff output, which is formatted strictly from left-to-right, finds occurrences of text in a right-to-left font and rearranges each line so that the text in each font is written in its proper direction. Additionally, ffortid left and right justifys lines containing Arabic & Persian fonts by stretching connections and/or letters in the words instead of inserting extra white space between the words in the lines.

1.4 Interface

command line options:

```
ffortid [ –rfont-position-list ] [ –wpaperwidth ] [ –afont-position-list ] ...
        [–s[n|[[l|c|e|b][f|2|m[amount]|a|ad|al]]] [–ms[c|l] ...
```

The **-r***font-position-list* argument is used to specify which font positions are to be considered right-to-left. The **-w***paperwidth* argument is used to specify the width of the paper, in inches, on which the the document will be printed. The **-a***font-position-list* argument is used to indicate which font positions, generally a subset of those designated as right-to-left (but not necessarily), contain fonts for Arabic, Persian or related languages. The **-s** argument specifies the kind of stretching to be done for all fonts designated in the **-a***font-position-list*. It is of the form **–s***mp* or **–sn** where *m* specifies the stretching mode, i.e, what is stretched, and *p* specifies the places that are stretched. The **–sn** means that there is *no* stretching (the default) and normal spreading of words is used even in **–a** designated fonts.

The choices for the mode *m* are:

l (letter ell) — In the words designated by the *p*, stretch the last stretchable letter.

c — In the words designated by the *p*, stretch the last connection to a letter.

e — In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later.

b — In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later, and if it is a letter that is stretched and it is a connect-previous letter then also stretch the connection to the letter.

1.4 Interface - Cont

The choices for the place *p* are:

`f` — In any line, stretch the last stretchable unit.

`2` — Assuming that the mode is **`b`** (both), in any line, stretch the last two stretchable units, if they are the connection leading to a stretchable connect-previous letter and that letter, and stretch only the last stretchable unit otherwise. If the mode is not **`b`**, then this choice of places is illegal.

`m`*n* or **`m`** — In any line, stretch the last stretchable unit by an amount not exceeding *n* emms. If that does not exhaust the available white space, then stretch the next last stretchable unit by an amount not exceeding *n* emms, and so on until all the available white space is exhausted. If *n* is not given, it is assumed to be **`2.0`**. In general *n* can be any number in floating point format.

`a`, **`ad`**, or **`al`** — In any line, stretch all stretchable units. In this case, the total amount available for stretching is divided evenly over all stretchable units on the line identified according to the mode. Since the units of stretching are the units of device resolution, the amount available might not divide evenly over the number of places. Therefore, it is useful to be able to specify what to do with the remainder of this division. This specification is given as an extension of the stretching argument. The choices are **`d`** or **`l`**, with the former indicating that the excess be distributed as evenly as possible to the spaces between words and the latter indicating that the excess be distributed as evenly as possible in stretchable letters that were stretchable units according to the current mode and place. The latter is the default if no choice is specified. The stretched item for the **`l`** choice must be a letter rather than a connection because only a stretchable letter is stretchable to any small amount that will be the remainder.

1.4 Interface - Cont

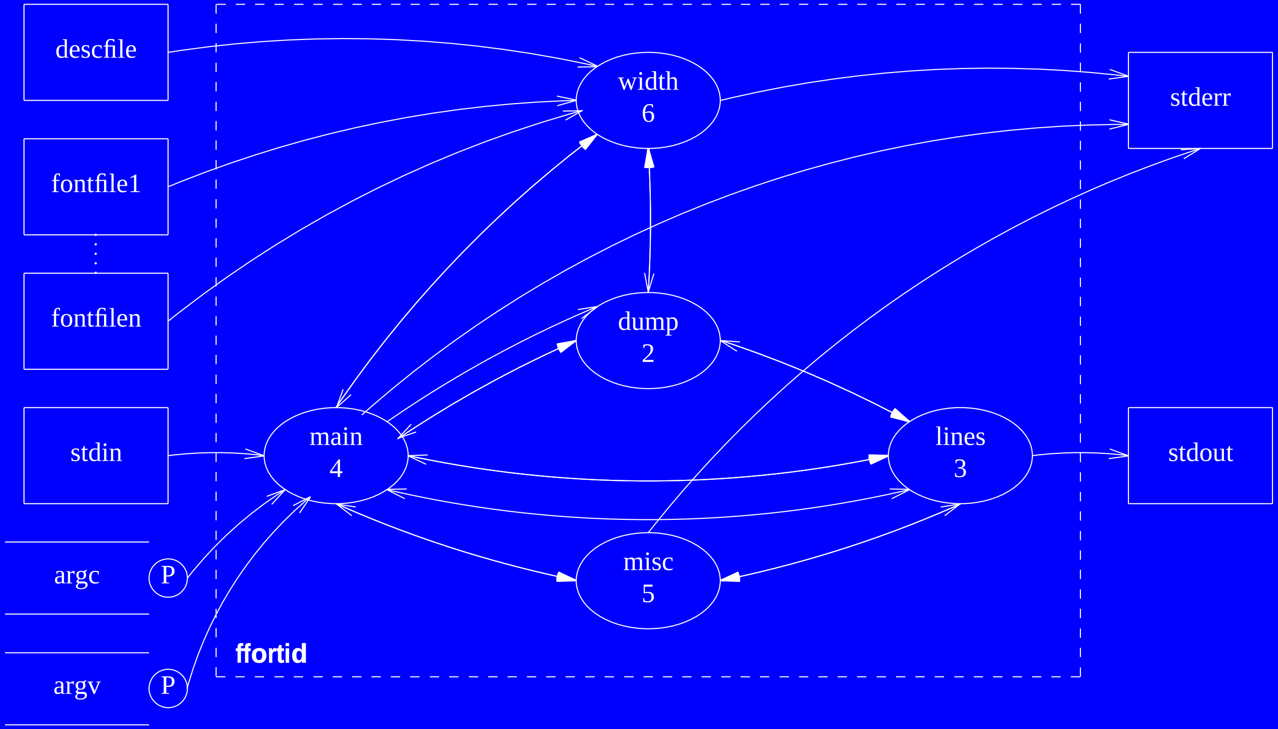
Manual connection stretching can be achieved by using explicitly the base-line filler character **`\(hy`** in the dtroff input. It can be repeated as many times as necessary to achieve the desired connection length.

Manual letter stretching can be achieved by immediately preceding, with no intervening white space, the letter to be stretched by **`\X'stretch'\h'`***n***`'`** where *n* is a valid length expression in troff's input language.

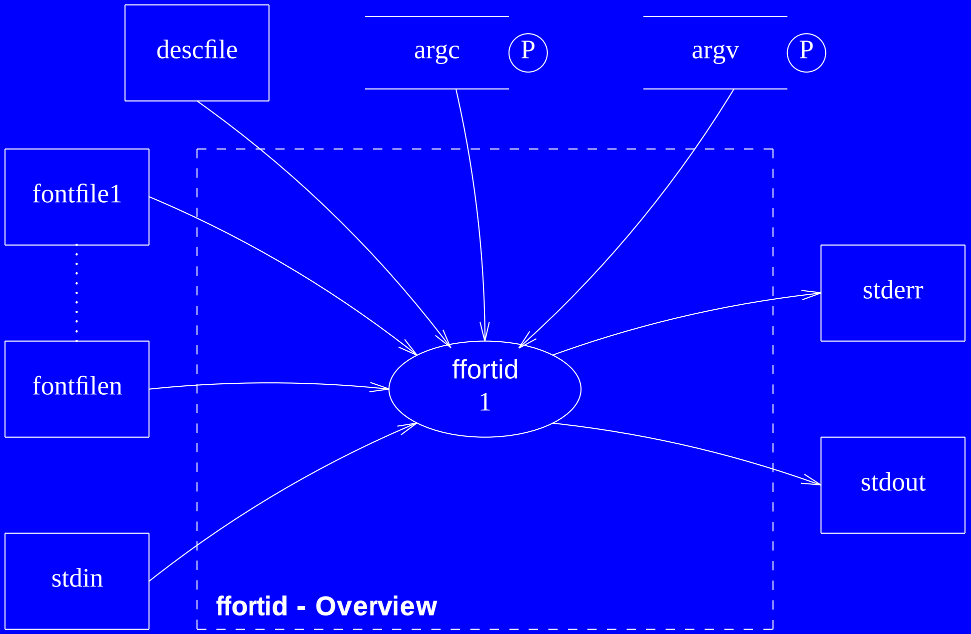
Finer control over automatic stretching of manually stretched connections and letters can be achieved by using the **–msc** and **–msl** flags. **–msc** prohibits automatic stretching of manually stretched connections. **–msl** prohibits automatic stretching of manually stretched letters.

Side effects:
1. ffortid reads dtroff output from stdin and prints dtroff output to stdout.
2. ffortid prints encountered errors to stderr and halts program.
3. ffortid allocates and frees memory from the heap. If out of heap memory ffortid prints a ``out of memory´´ message to stderr and halts program.

1.5 Service Flow Diagram



1.6 Service Flow Overview Diagram

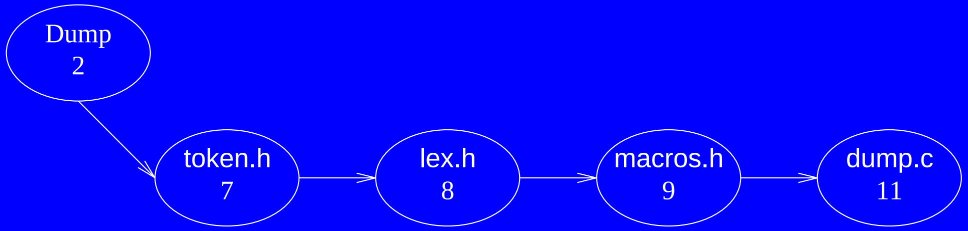


Software Unit #2 — Dump

2.1 Software Unit Type

Module. (token.h, lex.h, macros.h, dump.c)

2.2 Scope Diagram



2.3 Capabilities

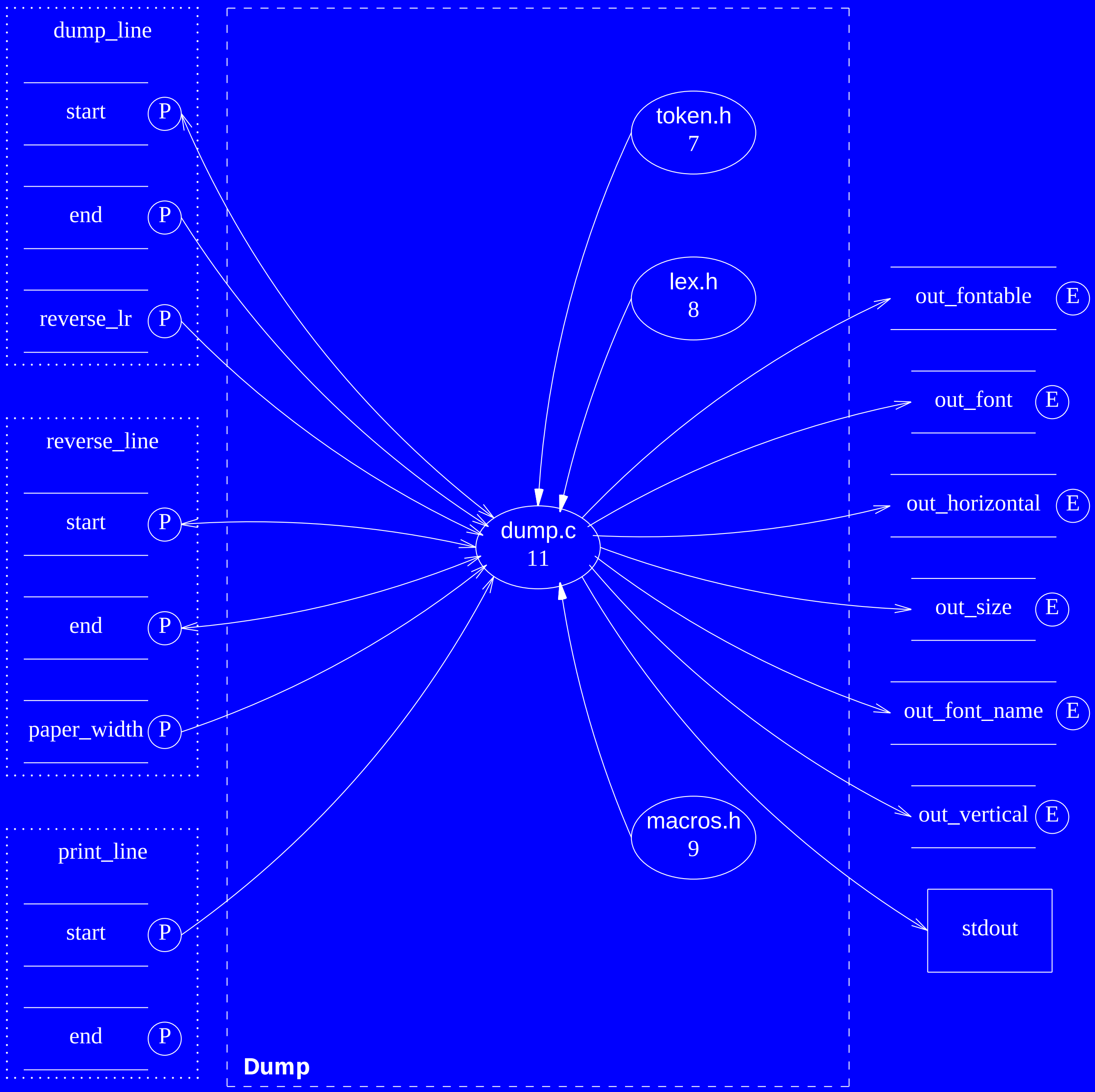
Contains routines that dump and reverse internal token lines while taking care of such issues as stretching and text direction.

2.4 Interface

Functions:
`dump_line` - stretches and dumps an internal token line while reversing tokens of the specified direction.
`reverse_line` - reverses an internal token line while preserving zero width characters position.
`print_line` - prints an internal token line to stdout. Used for debugging.

Side effects:
1. **`dump_line`** prints passed token line to stdout and frees the heap memory used by it.
2. **`dump_line`** changes the values of external vars: **`out_fontable`**, **`out_font`**, **`out_horizontal`**, **`out_size`**, **`out_font_name`**, **`out_vertical`**.
3. **`reverse_line`** changes the tokens in the passed token line.
4. **`print_line`** prints the passed token line to stdout.

2.5 Service Flow Diagram

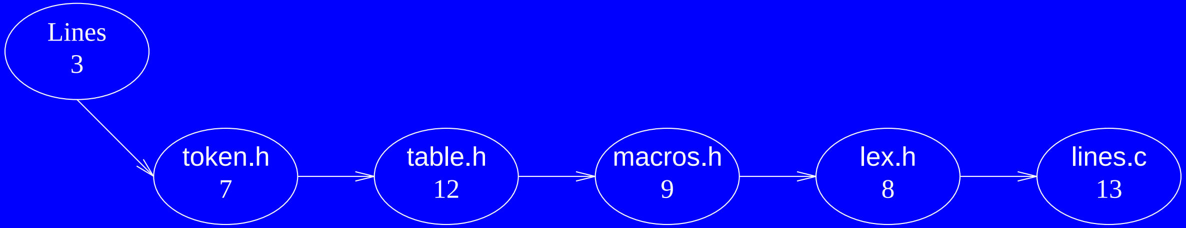


Software Unit #3 — Lines

3.1 Software Unit Type

Module. (token.h, table.h, macros.h, lex.h, lines.c)

3.2 Scope Diagram



3.3 Capabilities

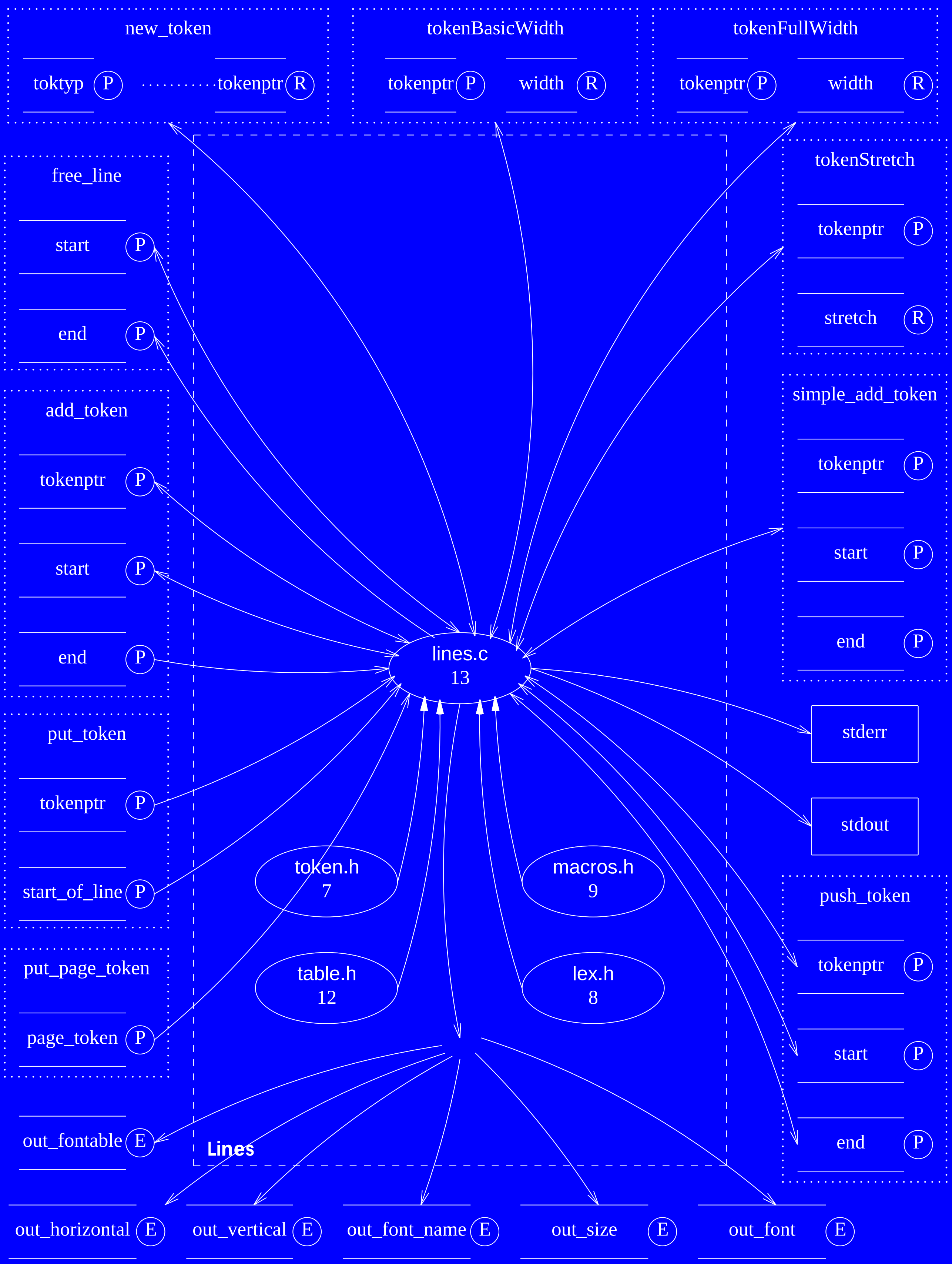
Contains routines to allocate, free, insert, print and inquire about width and stretch of internal tokens.

3.4 Interface

Functions:
`new_token` - allocates, initializes and returns a new internal token.
`free_line` - frees the memory allocated to a line of tokens.
`tokenBasicWidth` - return tokens basic width before stretching.
`tokenFullWidth` - return tokens full width after stretching.
`tokenStretch` - return tokens total stretch amount.
`add_token` - adds a token to the end of a line.
`simple_add_token` - adds a token to the end of a line without changing **`tokenptr`**.
`push_token` - pushes a token onto the front of a line.
`put_token` - outputs an internal token to stdout.
`put_page_token` - outputs a new page token and causes next **`put_token`** to print font & point sizes.

Side effects:
1. **`new_token`** allocates memory from the heap. If memory allocation fails then an ``out of memory´´ message is printed to stderr and the program halts.
2. **`free_line`** frees allocated memory to the heap.
3. **`add_token`**, **`simple_add_token`** and **`push_token`** change the passed token line.
4. **`put_token`** and **`put_page_token`** print tokens to stdout.
5. **`put_token`** changes the following external variables: **`out_fontable`**, **`out_font`**, **`out_horizontal`**, **`out_size`**, **`out_font_name`**, **`out_vertical`**.
6. **`put_page_token`** changes the following external variables: **`out_size`**, **`out_font_name`**, **`out_vertical`**.

3.5 Service Flow Diagram

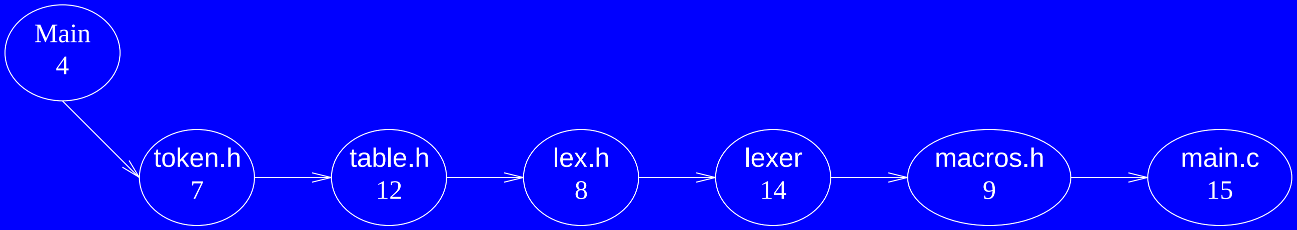


Software Unit #4 — Main

4.1 Software Unit Type

Module. (token.h, table.h, lex.h, lex.dit, macros.h, main.c)

4.2 Scope Diagram



4.3 Capabilities

Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

4.4 Interface

Globals:
`in_font` - current input font.
`in_size` - current input point size.
`in_horizontal` - current input horizontal position.
`in_vertical` - current input vertical position.
`in_font_name` - current input font name.
`in_lr` - current input font direction.
`in_fontable` - current input font table.
`out_font` - current output font.
`out_size` - current output point size.
`out_horizontal` - current output horizontal position.
`out_vertical` - current output vertical position.
`out_font_name` - current output font name.
`out_lr` - current output font direction.
`out_fontable` - current output font table.
`direction_table` - formatting direction of fonts table.
`arabic fonts` - boolean table stating which font is arabic.
`stretch_mode` - the stretching mode.
`stretch_place` - the stretching place.
`stretch_amount` - the stretch amount in emms.
`msc_flag` - manually stretched connections control flag.
`msl_flag` - manually stretched letters control flag.
`device` - name of output device.
`c` - general use char for flushing postscript and psfig text.

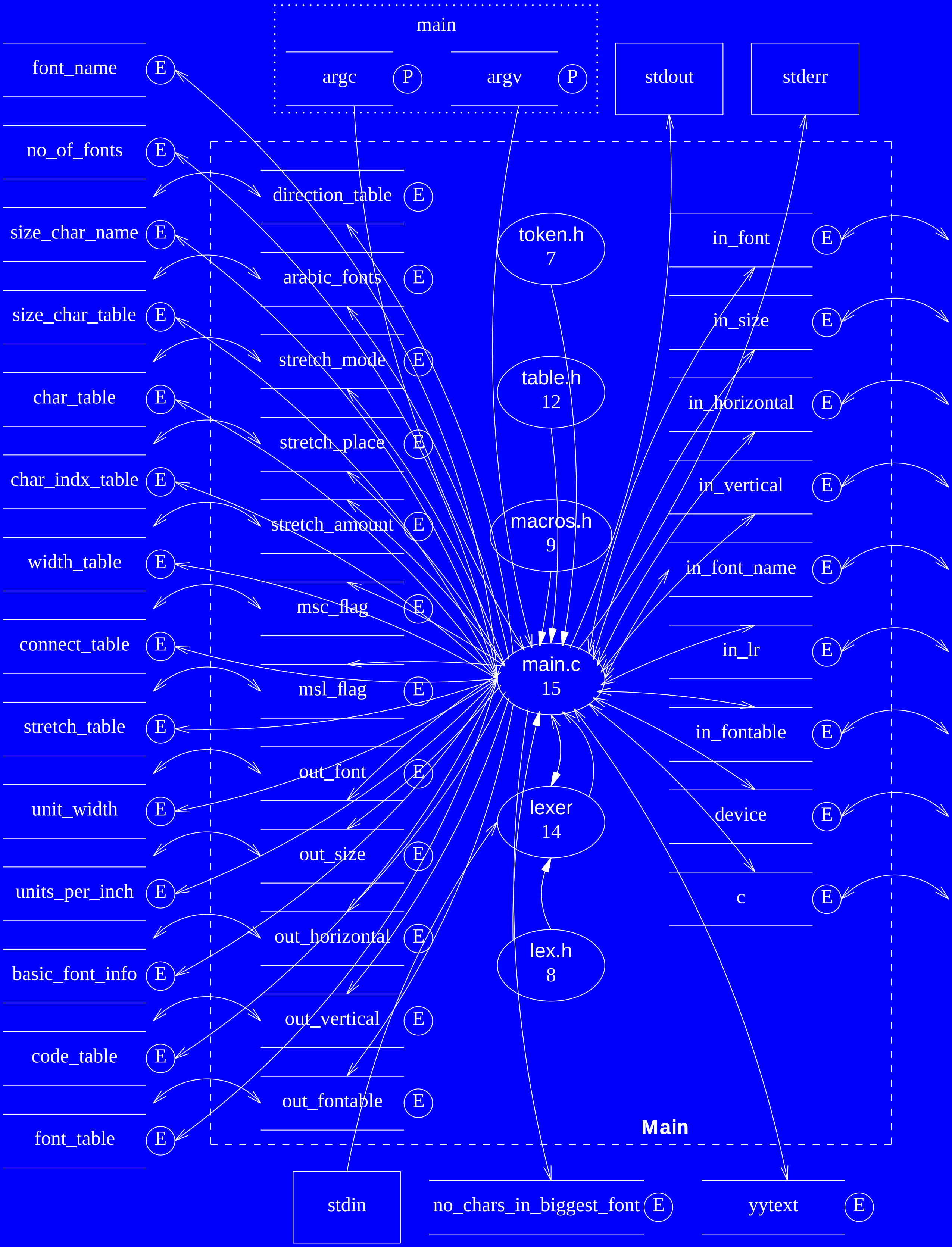
4.4 Interface - Cont

Functions:
`main` - main function for complete program including ffortid main driver.

Side effects:

1. **`main`** reads dtroff output from stdin and prints dtroff output to stdout.
2. **`main`** prints encountered errors to stderr and halts program.
3. **`main`** allocates and frees memory from the heap. If out of heap memory **`main`** prints a ``out of memory´´ message to stderr and halts program.
4. **`main`** changes the following external variables: **`font_name`**, **`no_of_fonts`**, **`size_char_name`**, **`size_char_table`**, **`char_table`**, **`char_indx_table`**, **`width_table`**, **`connect_table`**, **`stretch_table`**, **`unit_width`**, **`units_per_inch`**, **`basic_font_info`**, **`code_table`**, **`font_table`**, **`no_chars_in_biggest_font`**, **`yytext`**.

4.5 Service Flow Diagram

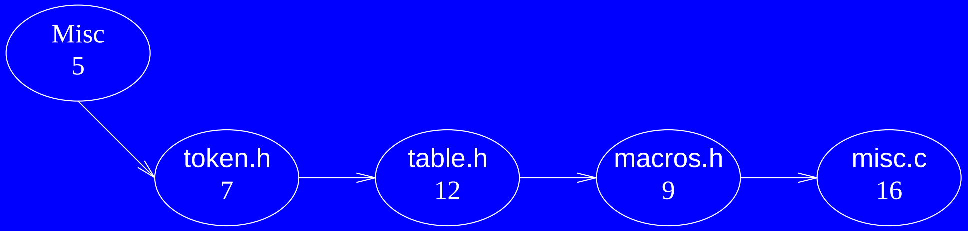


Software Unit #5 — Misc

5.1 Software Unit Type

Module (token.h, table.h, macros.h, misc.c)

5.2 Scope Diagram



5.3 Capabilities

Contains a number of general support routines.

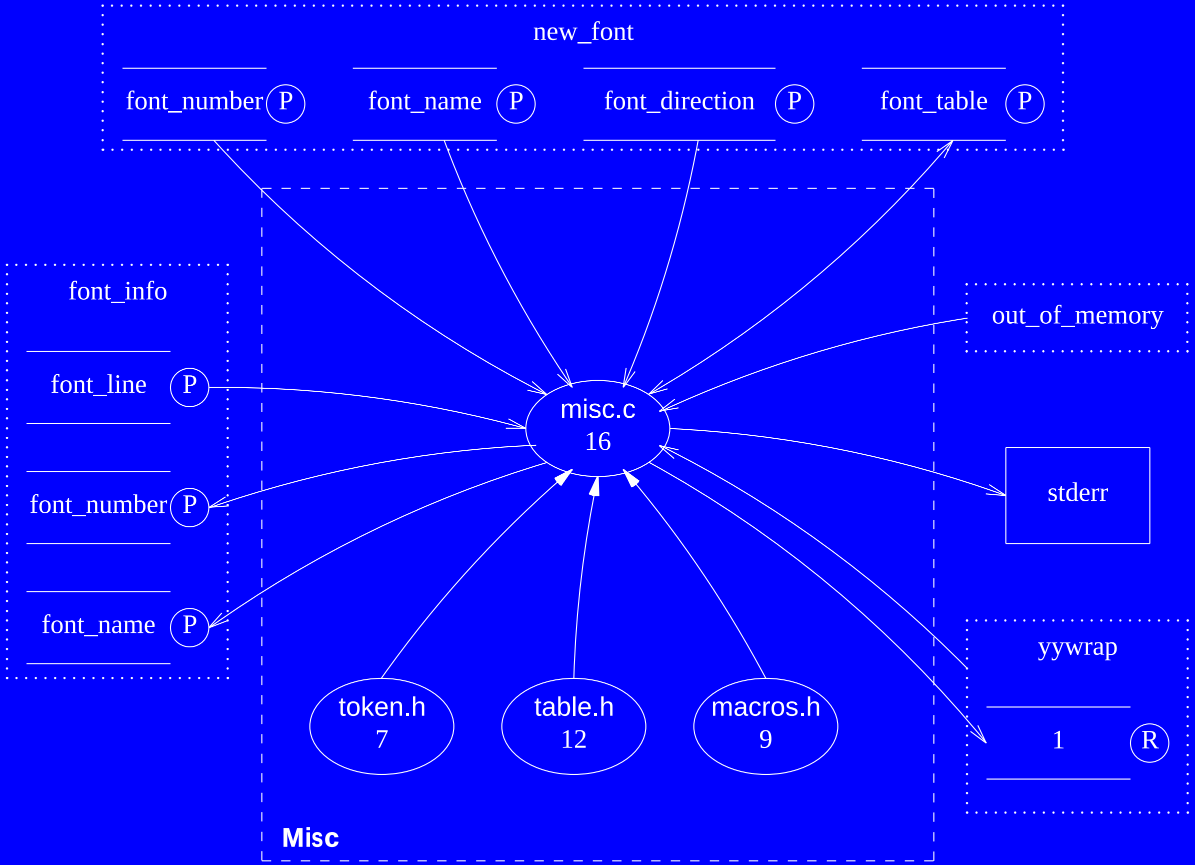
5.4 Interface

Functions:
`new_font` - adds a new font to the font table.
`font_info` - extracts a font number and name from a font token string.
`out_of_memory` - prints an ``out of memory´´ error message and halts execution.
`yywrap` - standard lex library function called whenever lex reaches an end-of-file.

Side effects:

1. **`new_font`** changes values in the passed **`font_table`**.
2. **`font_info`** returns through **`font_number`** the font token number and through**`font_name`** the font token name.
3. **`out_of_memory`** prints ``out of memory´´ error message to stderr and causes program to halt.

5.5 Service Flow Diagram

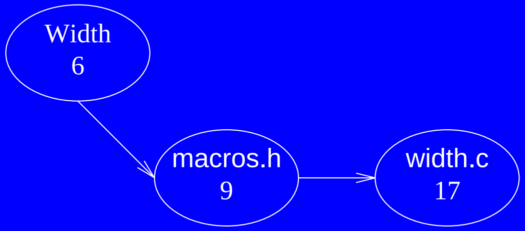


Software Unit #6 — Width

6.1 Software Unit Type

Module (width.c)

6.2 Scope Diagram



6.3 Capabilities

Contains globals that store the device and font width tables and routines to initialize them and return character widths, stretchability and connectivity based on them.

6.4 Interface

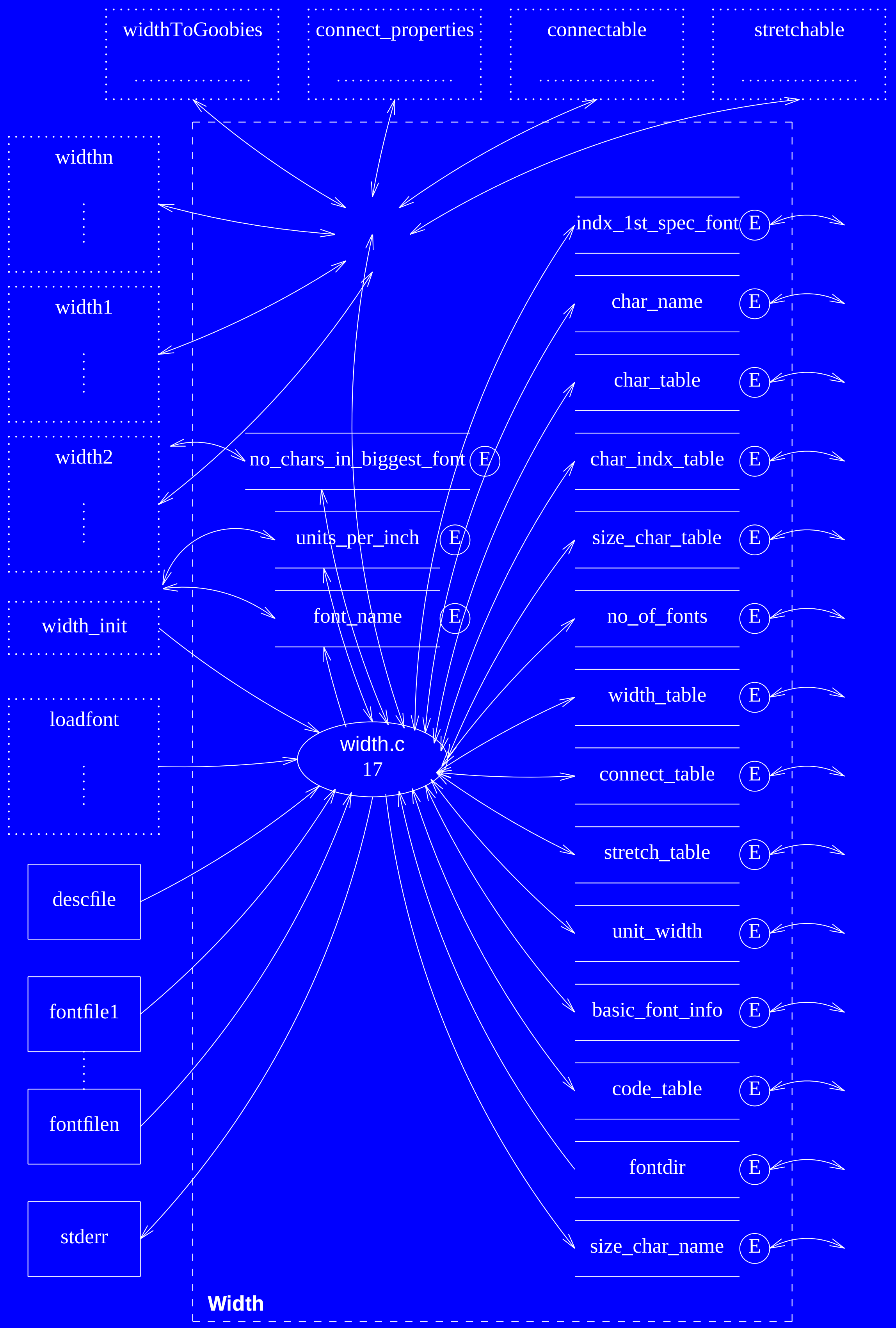
Globals:
`basic_font_info` - array of all fonts information.
`font_name` - array of all font names.
`no_of_fonts` - number of fonts initially mounted on the device.
`indx_1st_spec_font` - index of first special font.
`size_char_table` - size of character table in device.
`unit_width` - basic unit width in device.
`units_per_inch` - number of units per inch in device.
`no_chars_in_biggest_font` - number of chars in biggest font in device.
`size_char_name` - size of character name in device.
`char_name` - array of all character names in device.
`char_table` - array of indexes of characters in char_name.
`char_indx_table` - array of indexes of ascii characters in each font.
`code_table` - array of number codes for each char in each font.
`width_table` - array of widths for each char in each font.
`connect_table` - array of connectivity info for each char in each font.
`stretch_table` - array of stretchability info for each char in each font.
`fontdir` - font files directory.

6.4 Interface - Cont

Functions:
`width_init` - initializes the device and font tables.
`loadfont` - loads a single font table. Currently body commented out.
`width2` - returns the width of a specified funny character.
`width1` - returns the width of a specified character.
`widthn` - returns the width of a character specified with its code.
`widthToGoobies` - returns a width at a certain point size in goobies.
`connect_properties` - returns the connectivity of absolute char **`n`**.
`connectable` - returns whether absolute char **`n`** is a connect previous letter.
`stretchable` - returns whether absolute char **`n`** is stretchable.

Side effects:
1. **`width_init`** allocates memory from the heap.
2. Any error found in **`width_init`** is printed to stderr and the program halts.

6.5 Service Flow Diagram

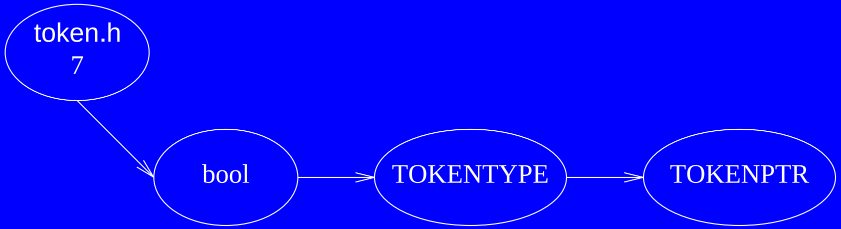


Software Unit #7 — token.h

7.1 Software Unit Type

Declerations source file. (token.h)

7.2 Scope Diagram



7.3 Capabilities

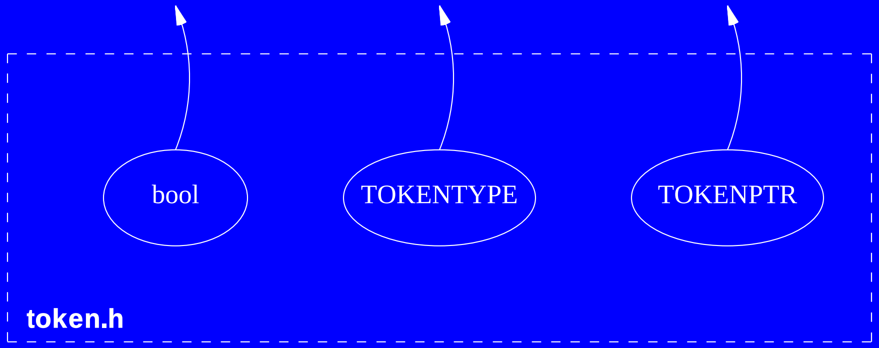
Contains the type declerations of the internal token representation structure and of **`bool`**.

7.4 Interface

Types:
`bool` - boolean values type.
`TOKENTYPE` - decleration of internal token representation structure.
`TOKENPTR` - decleration of pointer to **`TOKENTYPE`**.

Side effects:
None.

7.5 Service Flow Diagram

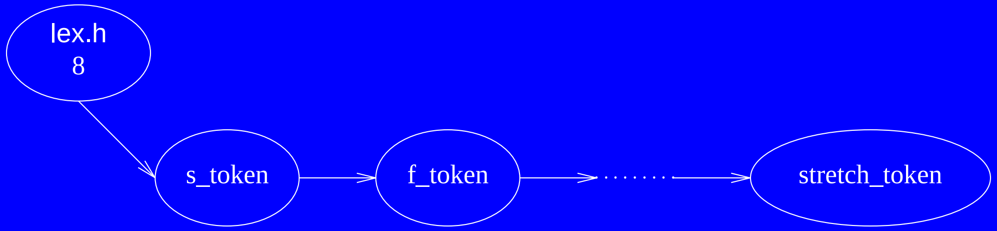


Software Unit #8 — lex.h

8.1 Software Unit Type

Definitions source file. (lex.h)

8.2 Scope Diagram



8.3 Capabilities

Contains 31 constant token definitions for lexical analyser.

8.4 Interface

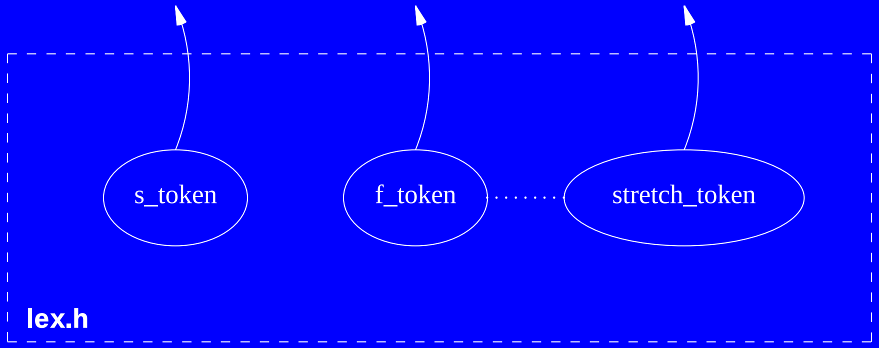
Constants:
`s_token` - dtroff s command token.
`f_token` - dtroff f command token.
`c_token` - dtroff c command token.
`C_token` - dtroff C command token.
`H_token` - dtroff H command token.
`V_token` - dtroff V command token.
`h_token` - dtroff h command token.
`v_token` - dtroff v command token.
`hc_token` - dtroff hc command token.
`n_token` - dtroff n command token.
`w_token` - dtroff w command token.
`p_token` - dtroff p command token.
`trail_token` - dtroff trail command token.
`stop_token` - dtroff stop command token.
`dev_token` - dtroff device command token.
`res_token` - dtroff resolution command token.
`init_token` - dtroff initialization command token.
`font_token` - dtroff font command token.
`pause_token` - dtroff pause command token.
`height_token` - dtroff height command token.
`slant_token` - dtroff slant command token.
`newline_token` - dtroff newline command token.
`PR_token` - dtroff page right-to-left command token.
`PL_token` - dtroff page left-to-right command token.

8.4 Interface - Cont

`D_token` - dtroff draw command token.
`N_token` - dtroff N command token.
`include_token` - dtroff include command token.
`control_token` - dtroff control command token.
`postscript_begin_token` - dtroff postscript begin command token.
`psfig_begin_token` - dtroff psfig begin command token.
`stretch_token` - dtroff manual stretch command token.

Side effects:
None.

8.5 Service Flow Diagram

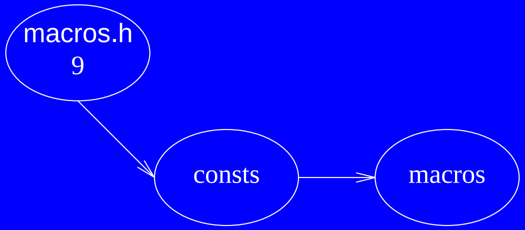


Software Unit #9 — macros.h

9.1 Software Unit Type

Definitions source file. (macros.h)

9.2 Scope Diagram



9.3 Capabilities

Contains general constant and macro definitions.

9.4 Interface

Constants:
`BEGINING` - token word begin constant.
`NOT_BEGIN` - token not word begin constant.
`LEFT_TO_RIGHT` - direction left to right constant.
`RIGHT_TO_LEFT` - direction right to left constant.
`END` - token word end constant.
`NOT_END` - token not word end constant.
`TRUE` - boolean true constant.
`FALSE` - boolean false constant.
`NOFILLERS` - token nofillers constant.
`NOSTRETCH` - token stretch constant.
`ARABIC` - font arabic constant.
`STRETCHABLE` - char stretchable constant.
`NOTSTRETCHABLE` - char not stretchable constant.
`STANDALONE` - char standalone constant.
`CONNECTAFTER` - char connect after constant.
`CONNECTPREVIOUS` - char connect previous constant.
`CONNECTBOTH` - char connected both constant.
`UNCONNECTED` - char unconnected constant.

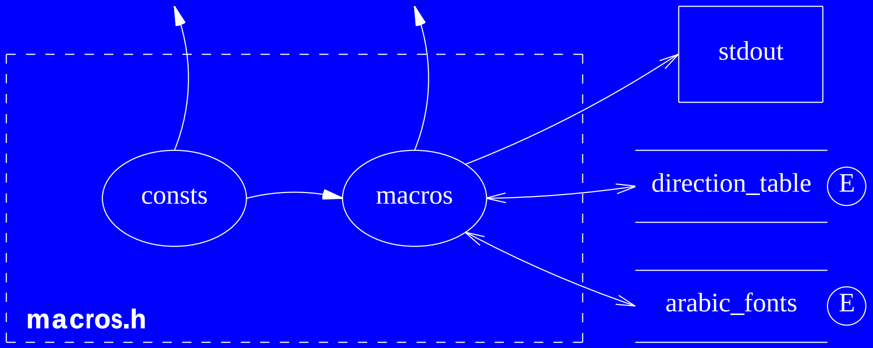
Macros:
`DUMP_LEX` - dump string to stdout as is.
`SET_DIRECTION` - set font direction.
`FONT_DIRECTION` - return font direction.

9.4 Interface - Cont

SET_AR_FONT - set font as arabic.
RESET_AR_FONT - set font as non-arabic.

Side effects:
1. **DUMP_LEX** prints to stdout.
2. **SET_DIRECTION** and **FONT_DIRECTION** change **direction_table**.
3. **SET_AR_FONT** and **RESET_AR_FONT** change **arabic_fonts**.

9.5 Service Flow Diagram

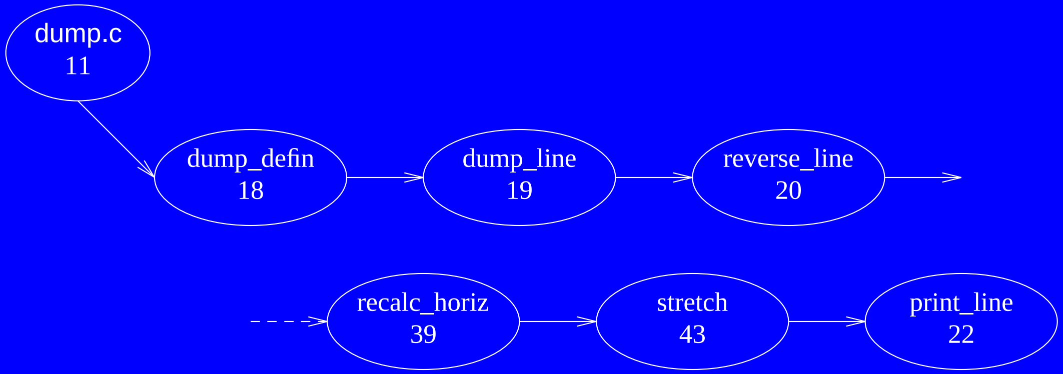


Software Unit #11 — dump.c

11.1 Software Unit Type

Source file. (dump.c)

11.2 Scope Diagram



11.3 Capabilities

Contains routines that dump and reverse internal token lines while taking care of such issues as stretching and text direction.

11.4 Interface

Constants:
`MAXZWC` - maximum number of respective zero width characters.

Macros:
`max` - maximum of two values.

Externals:
`arabic fonts` - boolean table stating which font is arabic.
`stretch_mode` - the stretching mode.
`stretch_place` - the stretching place.
`stretch_amount` - the stretch amount in emms.
`msc_flag` - manually stretched connections control flag.
`msl_flag` - manually stretched letters control flag.
`new_token()` - allocates, initializes and returns a new internal token.

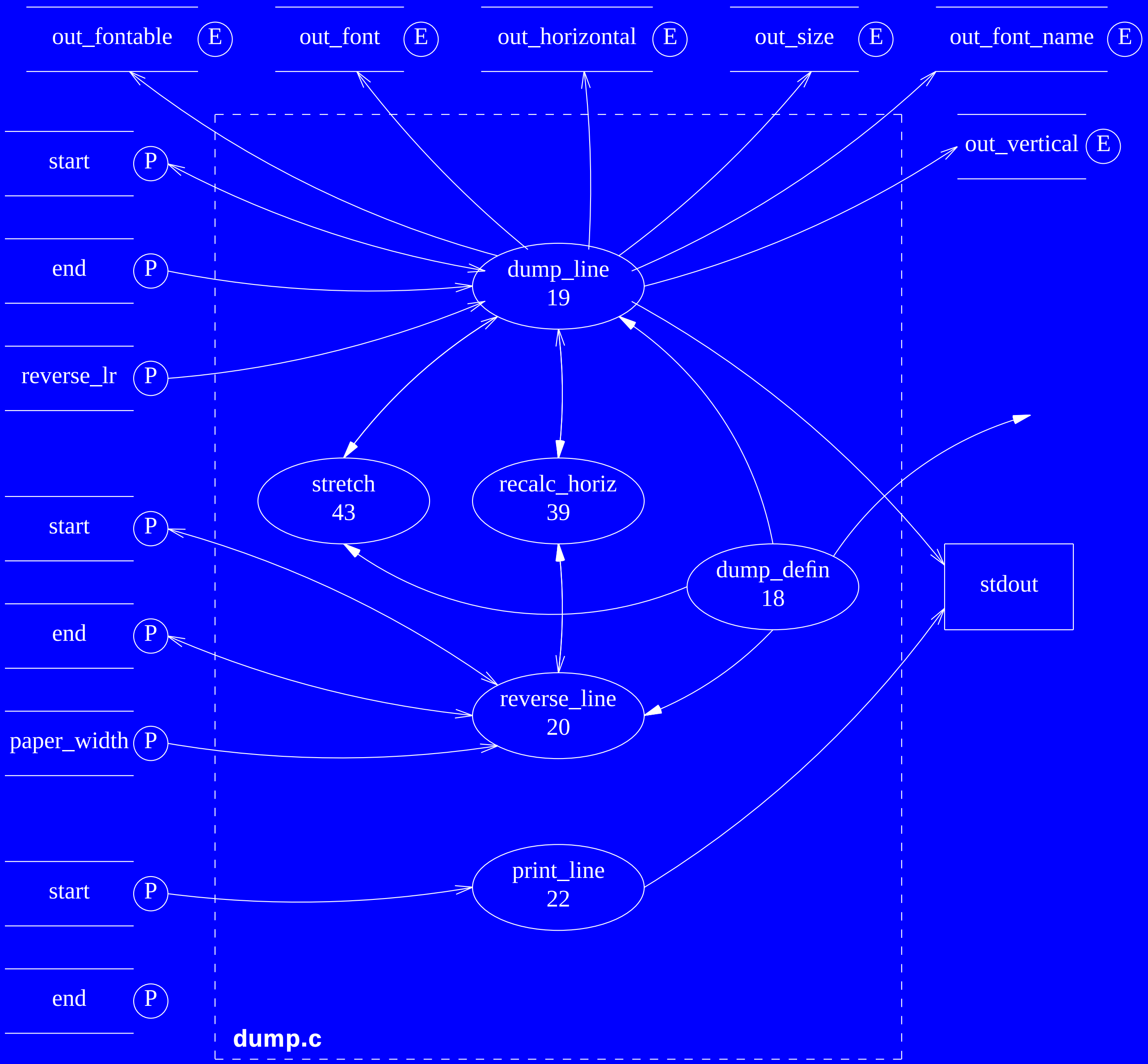
Functions:
`dump_line` - stretches and dumps an internal token line while reversing tokens of the specified direction.
`reverse_line` - reverses an internal token line while preserving zero width characters position.
`print_line` - prints an internal token line to stdout. Used for debugging.

11.4 Interface - Cont

Side effects:

1. **`dump_line`** prints passed token line to stdout and frees the heap memory used by it.
2. **`dump_line`** changes the values of external vars: **`out_fontable`**, **`out_font`**, **`out_horizontal`**, **`out_size`**, **`out_font_name`**, **`out_vertical`**.
3. **`reverse_line`** changes the tokens in the passed token line.
4. **`print_line`** prints the passed token line to stdout.

11.5 Service Flow Diagram

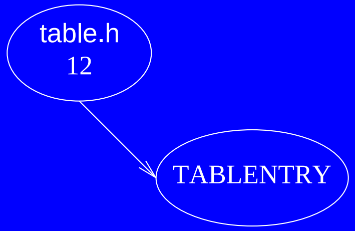


Software Unit #12 — table.h

12.1 Software Unit Type

Declerations source file. (table.h)

12.2 Scope Diagram



12.3 Capabilities

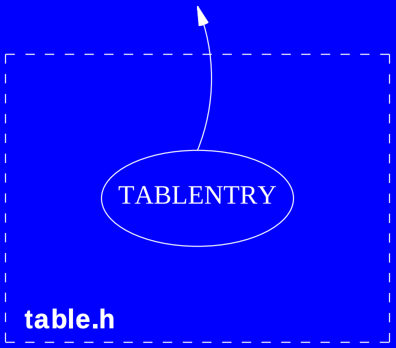
Contains the type decleration of the internal font table entry structure.

12.4 Interface

Types:
TABLENTRY - internal font table entry structure.

Side effects:
None.

12.5 Service Flow Diagram

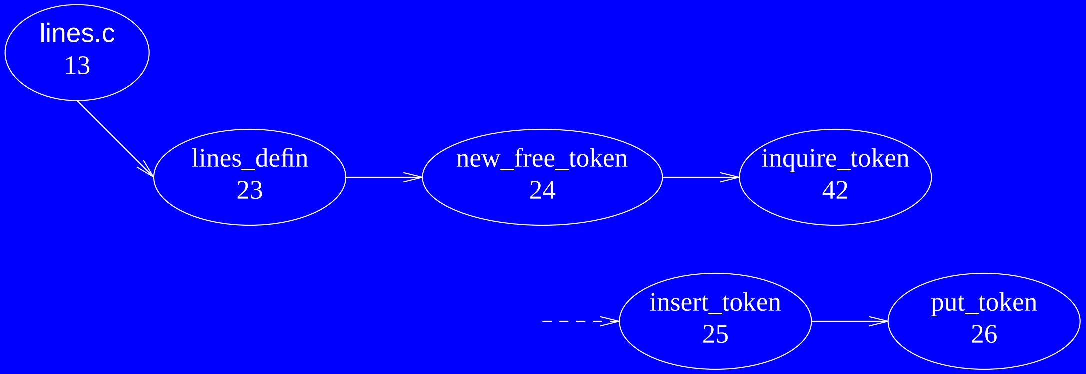


Software Unit #13 — lines.c

13.1 Software Unit Type

Source file. (lines.c)

13.2 Scope Diagram



13.3 Capabilities

Contains routines to allocate, free, insert, print and inquire about width and stretch of internal tokens.

13.4 Interface

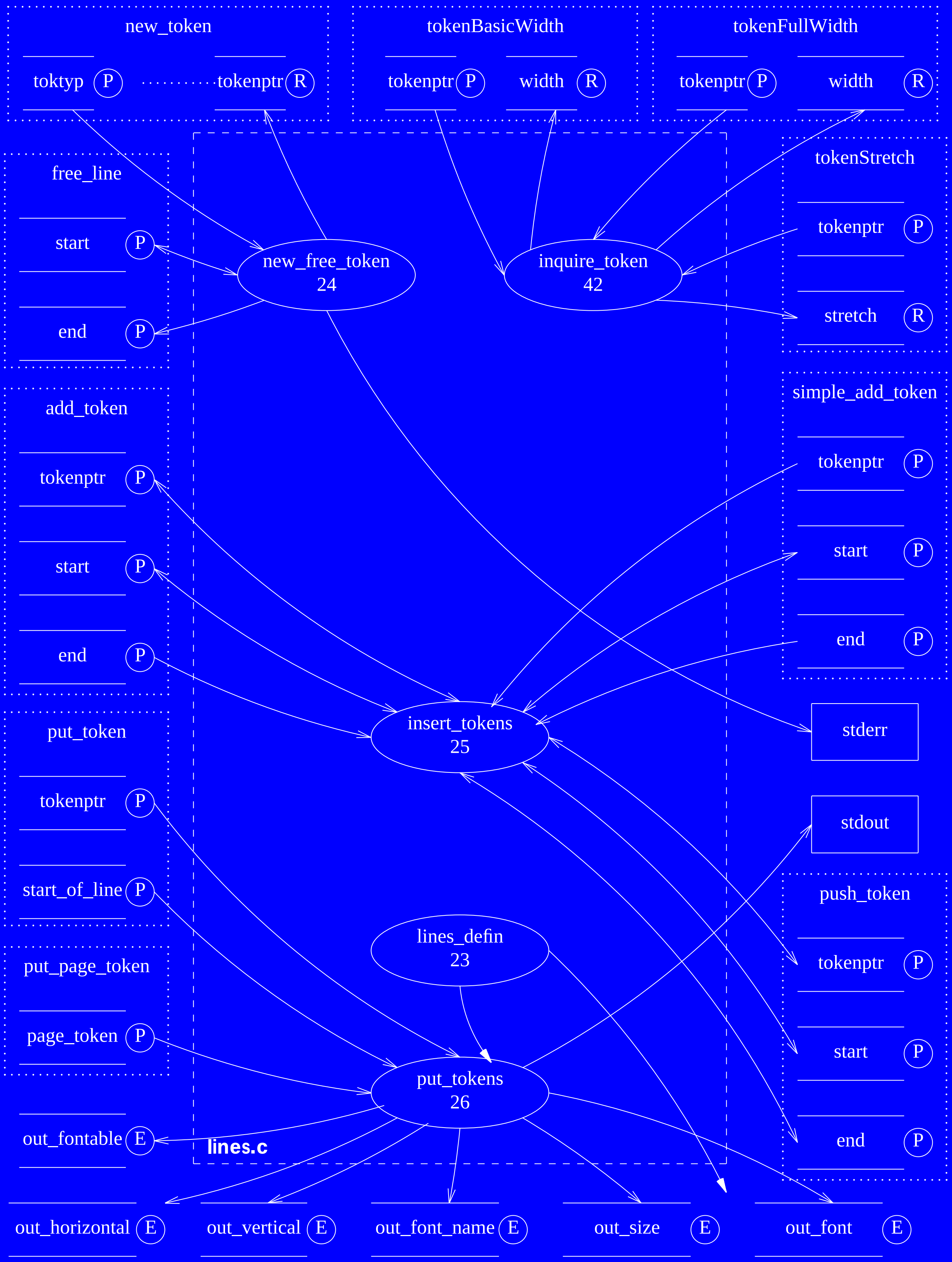
Externals:
`out_font` - current output font.
`out_size` - current output point size.
`out_horizontal` - current output horizontal position.
`out_vertical` - current output vertical position.
`out_font_name` - current output font name.
`out_fontable` - current output font table.
`in_font` - current input font.
`in_size` - current input point size.
`in_horizontal` - current input horizontal position.
`in_vertical` - current input vertical position.
`in_font_name` - current input font name.
`in_fontable` - current input font table.
`in_lr` - current input font direction.
`direction_table` - table of fonts formatting direction.

13.4 Interface - Cont

Functions:
new_token - allocates, initializes and returns a new internal token.
free_line - frees the memory allocated to a line of tokens.
tokenBasicWidth - return tokens basic width before stretching.
tokenFullWidth - return tokens full width after stretching.
tokenStretch - return tokens total stretch amount.
add_token - adds a token to the end of a line.
simple_add_token - adds a token to the end of a line without changing **tokenptr**.
push_token - pushes a token onto the front of a line.
put_token - outputs an internal token to stdout.
put_page_token - outputs a new page token and causes next **put_token** call to print font and point sizes.

Side effects:
1. **new_token** allocates memory from the heap. If memory allocation fails then an ``out of memory´´ message is printed to stderr and the program halts.
2. **free_line** frees allocated memory to the heap.
3. **add_token**, **simple_add_token** and **push_token** change the passed token line.
4. **put_token** and **put_page_token** print tokens to stdout.
5. **put_token** changes the following external variables: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
6. **put_page_token** changes the following external variables: **out_size**, **out_font_name**, **out_vertical**.

13.5 Service Flow Diagram

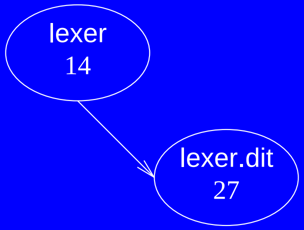


Software Unit #14 — lexer

14.1 Software Unit Type

Lex generated source file. (lex.dit)

14.2 Scope Diagram



14.3 Capabilities

Lexicaly parses dtroff output into tokens.

14.4 Interface

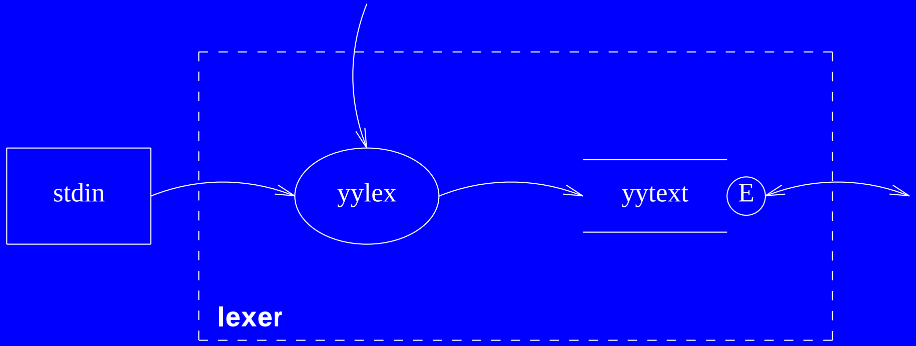
Globals:
`yytext` - points to the actual string matched by the lexical analyser.

14.4 Interface - Cont

Functions:
`yylex` - returns next token matched by the lexical analyser.

Side effects:
Reads in dtroff output from stdin.

14.5 Service Flow Diagram

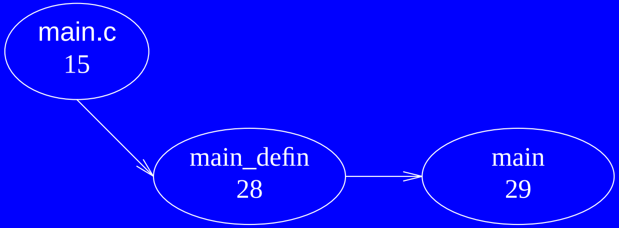


Software Unit #15 — main.c

15.1 Software Unit Type

Source file. (main.c)

15.2 Scope Diagram



15.3 Capabilities

Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

15.4 Interface

Constants:
`USAGE` - command line usage explanation string.

Macros:
`MARK_PREVIOUS_END` - marks the last token in the current input line as ending a word.
`ADD_CHAR1` - creates a new token from 1 char and adds it to end of current input line.
`ADD_CHAR2` - creates a new token from 2 chars and adds it to end of current input line.
`ADD_CHARN` - creates a new token of from 3 chars and adds it to end of current input line.

Static Globals:
`copyright` - string holding copyright information.

Globals:
`in_font` - current input font.
`in_size` - current input point size.
`in_horizontal` - current input horizontal position.
`in_vertical` - current input vertical position.
`in_font_name` - current input font name.
`in_lr` - current input font direction.
`in_fontable` - current input font table.
`out_font` - current output font.
`out_size` - current output point size.
`out_horizontal` - current output horizontal position.
`out_vertical` - current output vertical position.
`out_font_name` - current output font name.

15.4 Interface - Cont

`out_lr` - current output font direction.
`out_fontable` - current output font table.
`direction_table` - formatting direction of fonts table.
`arabic fonts` - boolean table stating which font is arabic.
`stretch_mode` - the stretching mode.
`stretch_place` - the stretching place.
`stretch_amount` - the stretch amount in emms.
`msc_flag` - manually stretched connections control flag.
`msl_flag` - manually stretched letters control flag.
`device` - name of output device.
`c` - general use char for flushing postscript and psfig text.

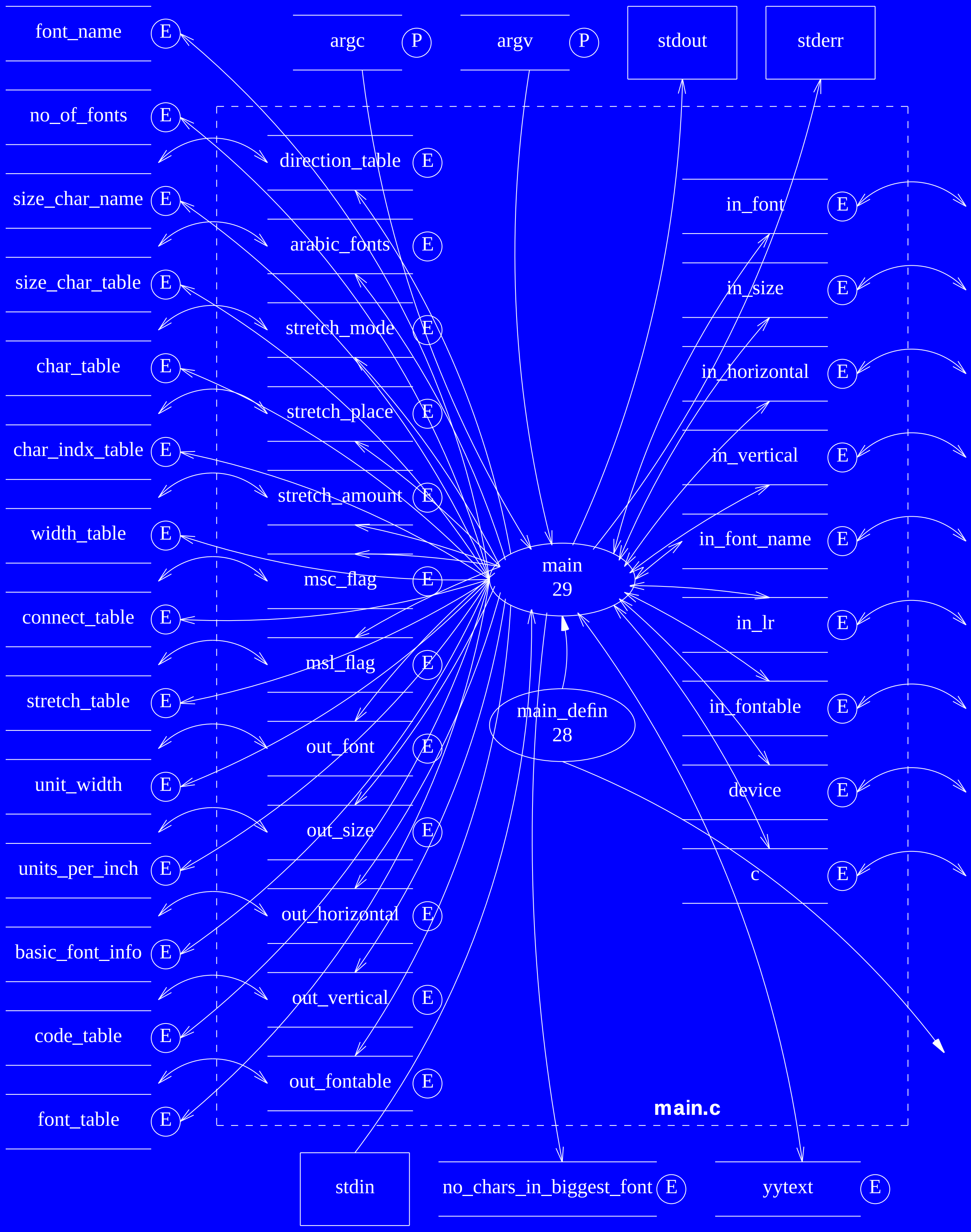
Functions:
`main` - main function for complete program including ffortid main driver.

15.4 Interface - Cont

Side effects:

1. **MARK_PREVIOUS_END** changes the token pointed by **in_end**.
2. **ADD_CHAR1**, **ADD_CHAR2** and **ADD_CHARN** create a new token allocated from the heap and add it to the token line pointed to by **in_start** and **in_end**.
3. **main** reads dtroff output from stdin and prints dtroff output to stdout.
4. **main** prints encountered errors to stderr and halts program.
5. **main** allocates and frees memory from the heap. If out of heap memory **main** prints a ``out of memory´´ message to stderr an halts program.
6. **main** changes the following external variables: **font_name**, **no_of_fonts**, **size_char_name**, **size_char_table**, **char_table**, **char_indx_table**, **width_table**, **connect_table**, **stretch_table**, **unit_width**, **units_per_inch**, **basic_font_info**, **code_table**, **font_table**, **no_chars_in_biggest_font**, **yytext**.

15.5 Service Flow Diagram

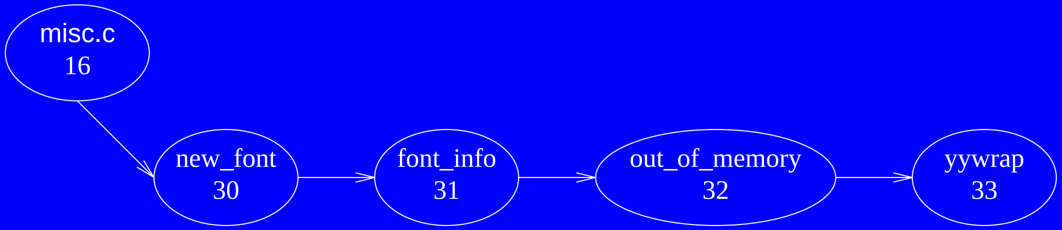


Software Unit #16 — misc.c

16.1 Software Unit Type

Source file. (misc.c)

16.2 Scope Diagram



16.3 Capabilities

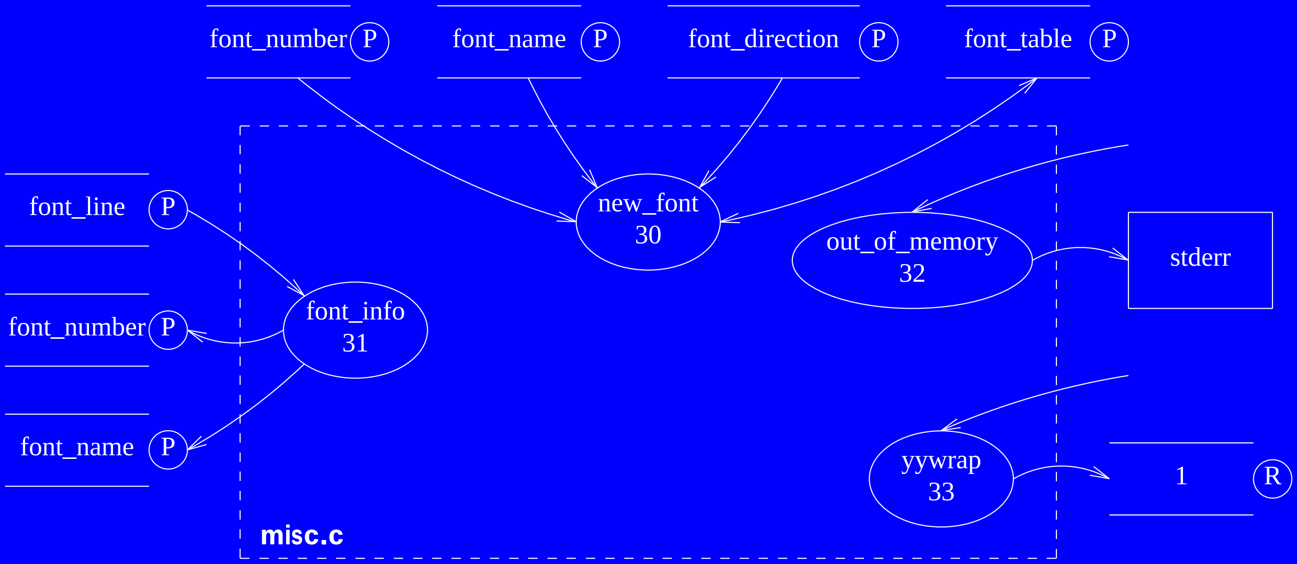
Contains a number of general support routines.

16.4 Interface

Functions:
`new_font` - adds a new font to the font table.
`font_info` - extracts a font number and name from a font token string.
`out_of_memory` - prints an ``out of memory´´ error message and halts execution.
`yywrap` - standard lex library function called whenever lex reaches an end-of-file.

Side effects:
1. **`new_font`** changes values in the passed **`font_table`**.
2. **`font_info`** returns through **`font_number`** the font token number and through **`font_name`** the font token name.
3. **`out_of_memory`** prints ``out of memory´´ error message to stderr and causes program to halt.

16.5 Service Flow Diagram

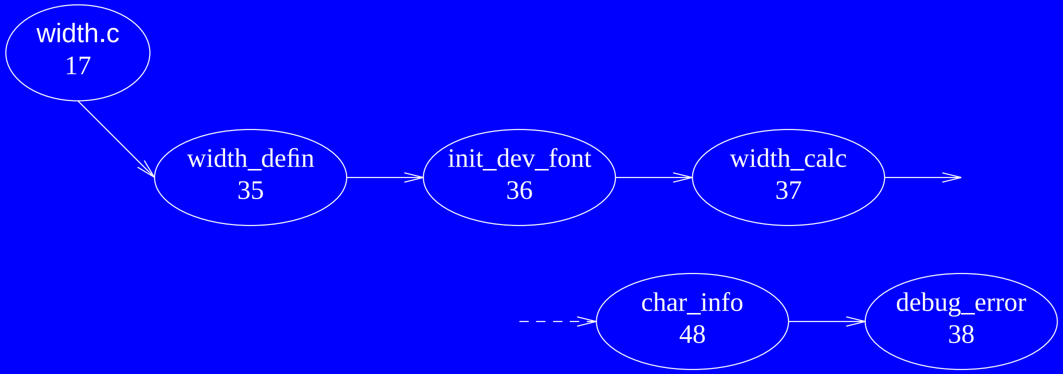


Software Unit #17 — width.c

17.1 Software Unit Type

Source file. (width.c)

17.2 Scope Diagram



17.3 Capabilities

Contains globals that store the device and font width tables and routines to initialize them and return character widths, stretchability and connectivity based on them.

17.4 Interface

Constants:
`MAXNOFONTS` - max number of fonts.
`MAXWIDENTRIES` - max width entries.
`NOCHARSINBIGGESTFONT` - no of characters in biggest font in device description.
`MAXNOCHARS` - max number of chars with with two letters or --- names.
`SIZECHARINDXTABLE` - size of character index table including ascii chars but not non-graphics.
`FATAL` - passed to error procedure to signal fatal error.
`BYTEMASK` - mask used to make character numbers positive.

Types:
`Fontinfo` - single font information structure.

Globals:
`basic_font_info` - array of all fonts information.
`font_name` - array of all font names.
`no_of_fonts` - number of fonts initially mounted on the device.
`indx_1st_spec_font` - index of first special font.
`size_char_table` - size of character table in device.
`unit_width` - basic unit width in device.
`units_per_inch` - number of units per inch in device.
`no_chars_in_biggest_font` - number of chars in biggest font in device.
`size_char_name` - size of character name in device.
`char_name` - array of all character names in device.
`char_table` - array of indexes of characters in char_name.
`char_indx_table` - array of indexes of ascii characters in each font.

17.4 Interface - Cont

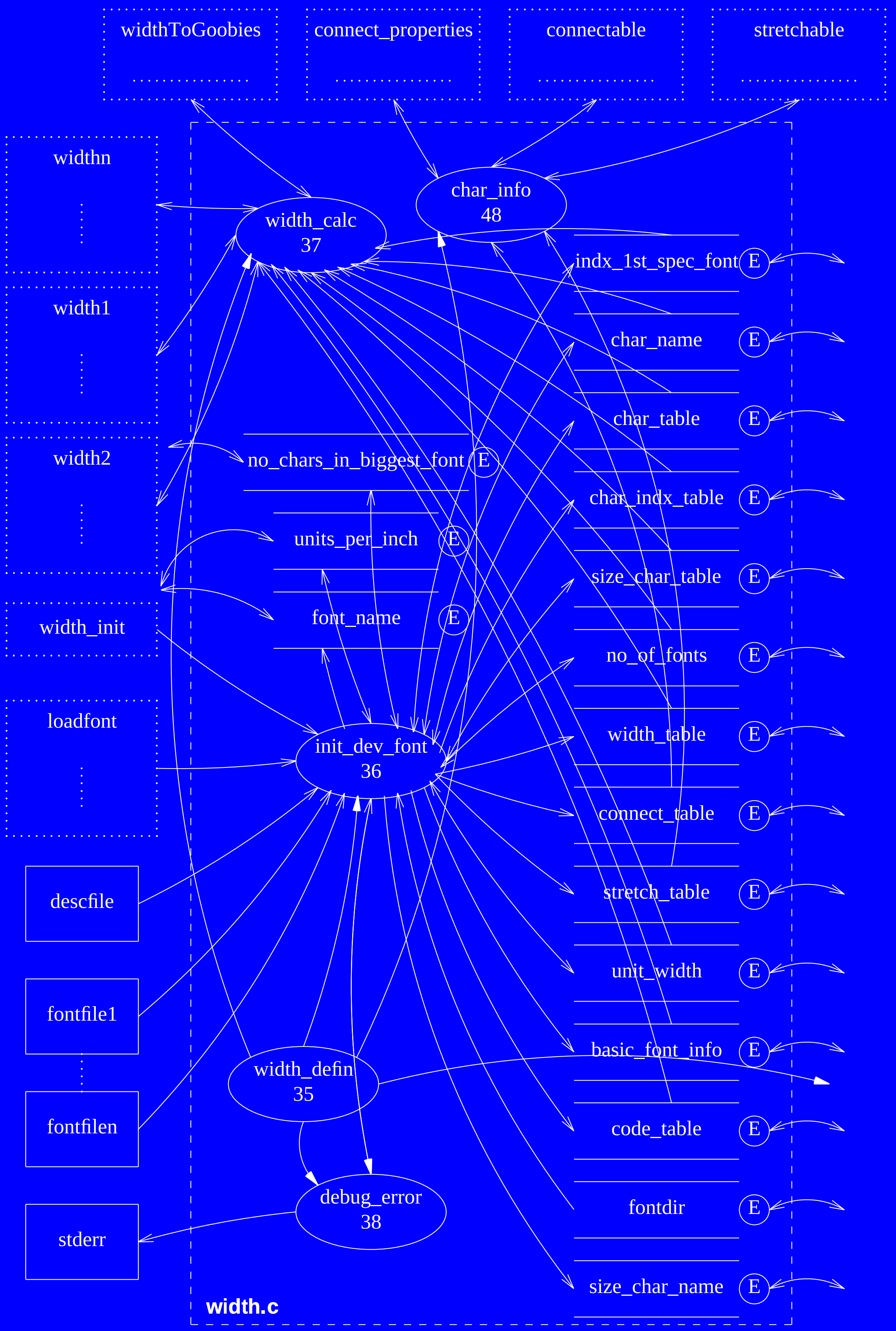
`code_table` - array of number codes for each char in each font.
`width_table` - array of widths for each char in each font.
`connect_table` - array of connectivity info for each char in each font.
`stretch_table` - array of stretchability info for each char in each font.
`fontdir` - font files directory.

Externals:
`device` - name of output device.

Functions:
`width_init` - initializes the device and font tables.
`loadfont` - loads a single font table. Currently body commented out.
`width2` - returns the width of a specified funny character.
`width1` - returns the width of a specified character.
`widthn` - returns the width of a character specified with its code.
`widthToGoobies` - returns a width at a certain point size in goobies.
`connect_properties` - returns the connectivity of absolute char **`n`**.
`connectable` - returns whether absolute char **`n`** is a connect previous letter.
`stretchable` - returns whether absolute char **`n`** is stretchable.

Side effects:
1. **`width_init`** allocates memory from the heap.
2. Any error found in **`width_init`** is printed to stderr and the program halts.

17.5 Service Flow Diagram

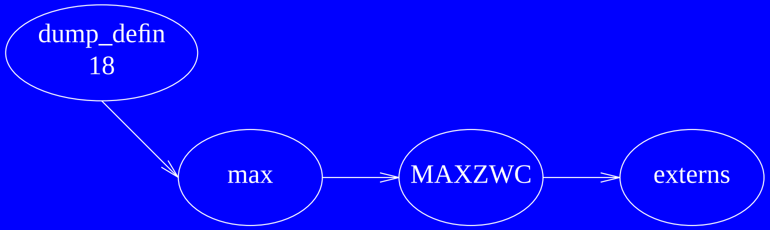


Software Unit #18 — dump_defin

18.1 Software Unit Type

Definitions block. (dump.c: 1-30)

18.2 Scope Diagram



18.3 Capabilities

Contains definitions used by the functions in dump.c.

18.4 Interface

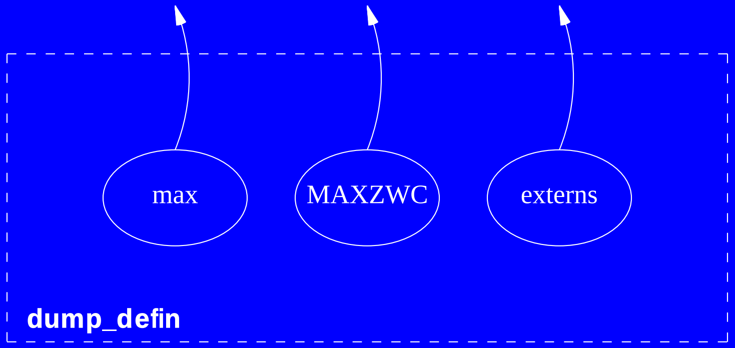
Constants:
`MAXZWC` - maximum number of respective zero width characters.

Macros:
`max` - maximum of two values.

Externals:
`arabic fonts` - boolean table stating which font is arabic.
`stretch_mode` - the stretching mode.
`stretch_place` - the stretching place.
`stretch_amount` - the stretch amount in emms.
`msc_flag` - manually stretched connections control flag.
`msl_flag` - manually stretched letters control flag.
`new_token()` - allocates, initializes and returns a new internal token.

Side effects:
None

18.5 Service Flow Diagram

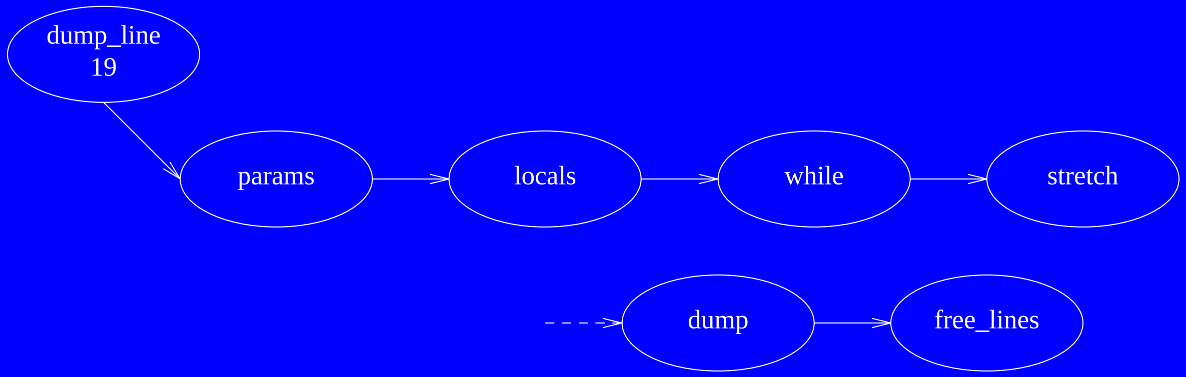


Software Unit #19 — dump_line

19.1 Software Unit Type

Procedure. (dump.c: 31-155)

19.2 Scope Diagram



19.3 Capabilities

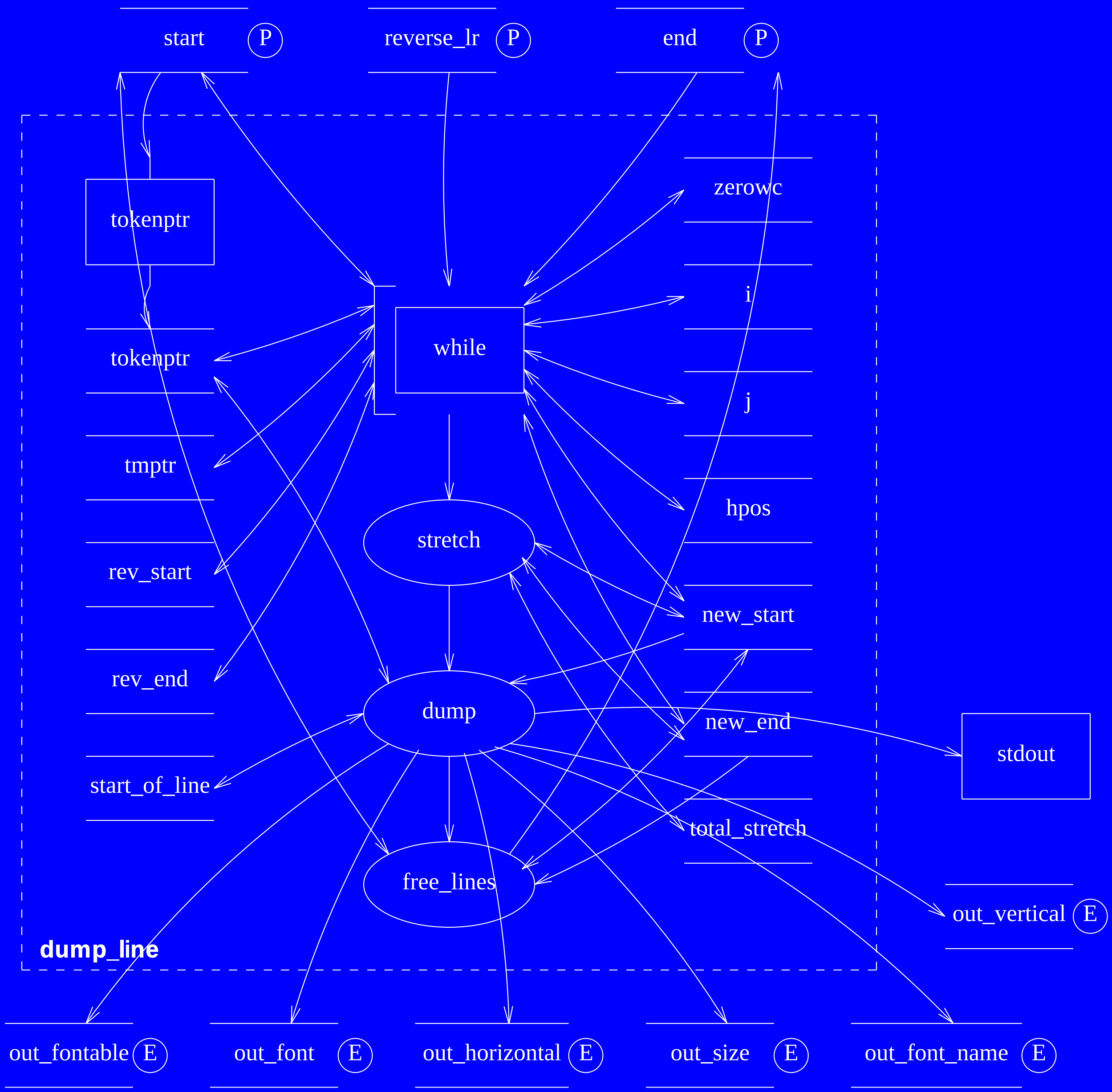
Stretches and dumps the specified internal token line to stdout while reversing the tokens of the specified direction. Deals also with zero width characters and zero horizontal movements.

19.4 Interface

Parameters:
`start` - pointer to first token in line.
`end` - pointer to last token in line.
`reverse_lr` - boolean specifying tokens of which direction are to be reversed.

Side effects:
1. Prints dumped line to stdout.
2. Changes the values of external vars: **`out_fontable`**, **`out_font`**, **`out_horizontal`**, **`out_size`**, **`out_font_name`**, **`out_vertical`**.
3. Frees the heap memory used by the passed token line.

19.5 Service Flow Diagram

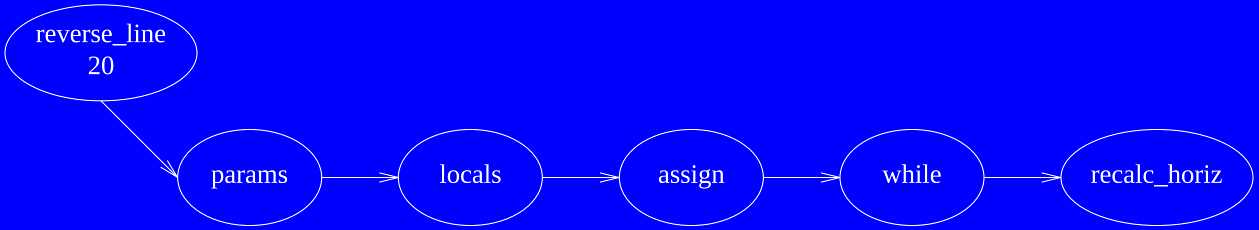


Software Unit #20 — reverse_line

20.1 Software Unit Type

Procedure. (dump.c: 156-242)

20.2 Scope Diagram



20.3 Capabilities

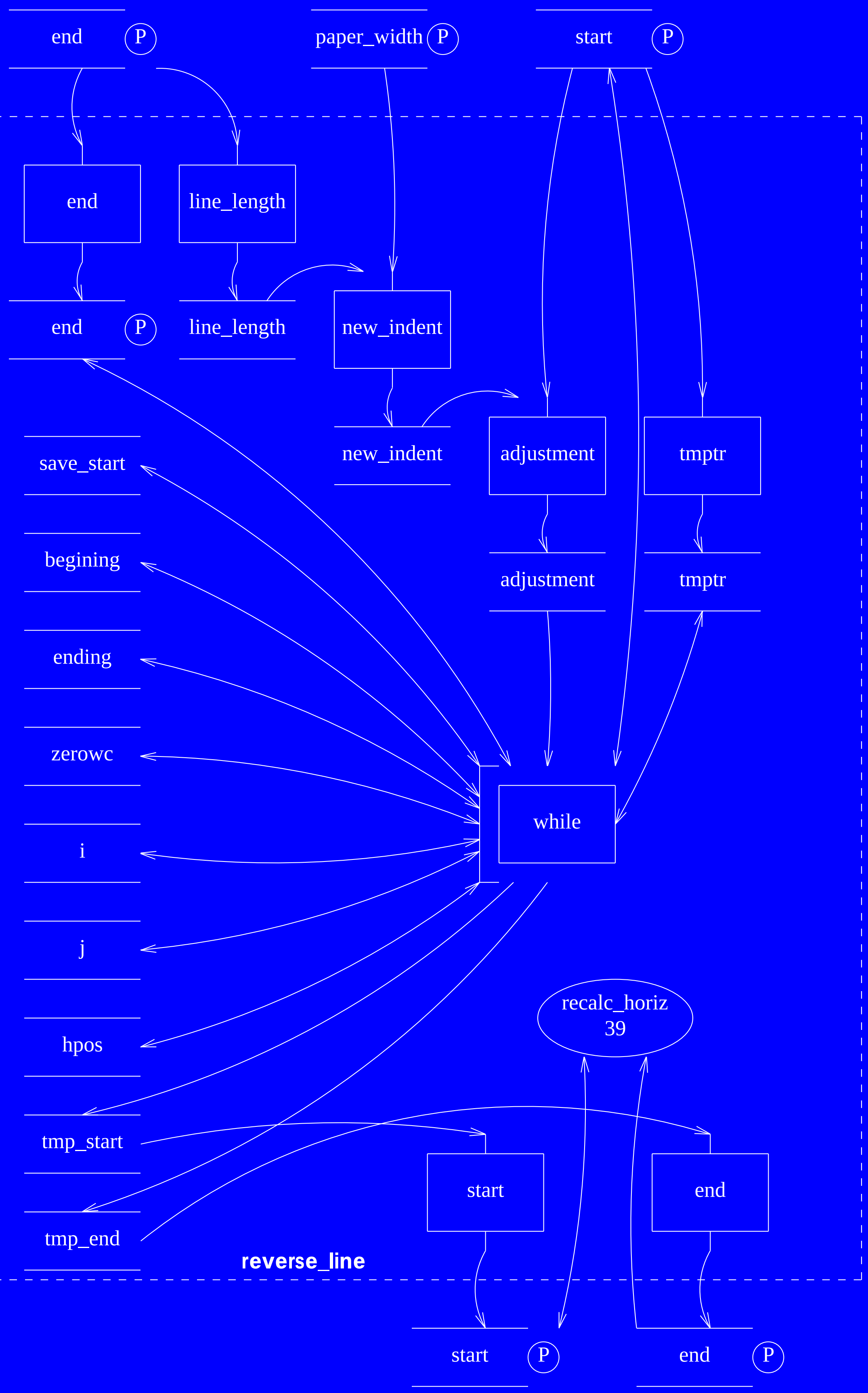
Reverses the specified internal token line while preserving the order of zero width characters with their next letter.

20.4 Interface

Parameters:
`start` - pointer to first token in line.
`end` - pointer to last token in line.
`paper_width` - paper width in points.

Side effects:
Changes the tokens in the passed token line.

20.5 Service Flow Diagram

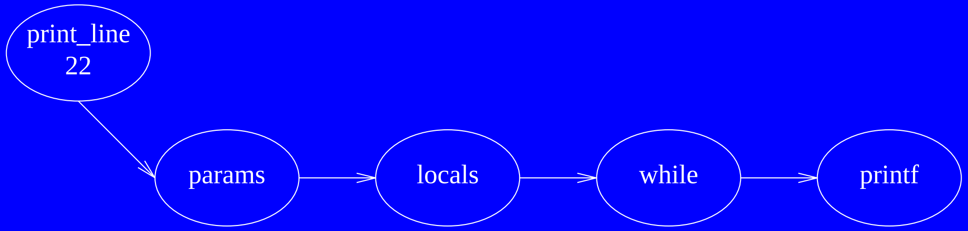


Software Unit #22 — print_line

22.1 Software Unit Type

Procedure. (dump.c: 897-917)

22.2 Scope Diagram



22.3 Capabilities

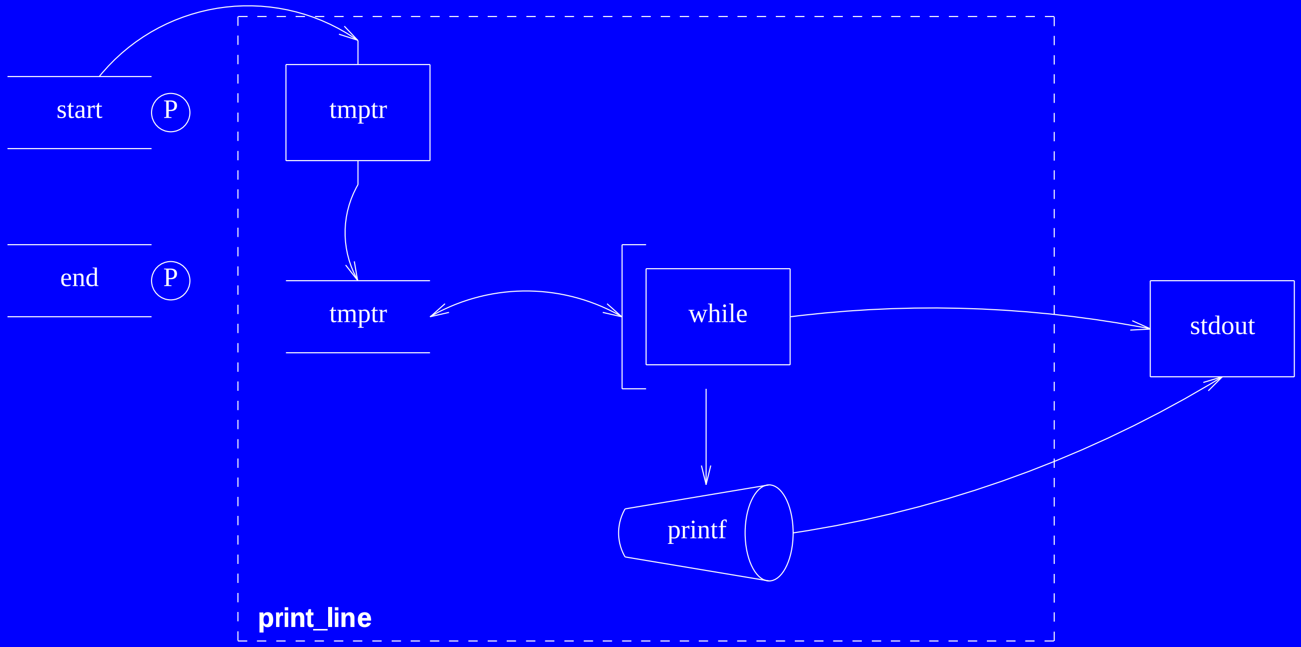
Prints the specified internal token line to stdout for debugging.

22.4 Interface

Parameters:
`start` - pointer to first token in line.
`end` - pointer to last token in line.

Side effects:
Prints the passed token line to stdout.

22.5 Service Flow Diagram

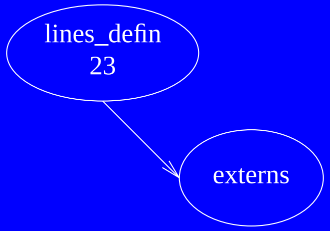


Software Unit #23 — lines_defin

23.1 Software Unit Type

Definitions block. (lines.c: 1-33)

23.2 Scope Diagram



23.3 Capabilities

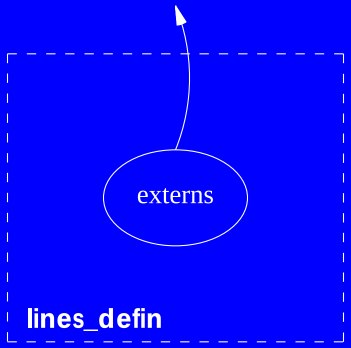
Contains definitions used by the lines.c functions.

23.4 Interface

Externals:
`out_font` - current output font.
`out_size` - current output point size.
`out_horizontal` - current output horizontal position.
`out_vertical` - current output vertical position.
`out_font_name` - current output font name.
`out_fontable` - current output font table.
`in_font` - current input font.
`in_size` - current input point size.
`in_horizontal` - current input horizontal position.
`in_vertical` - current input vertical position.
`in_font_name` - current input font name.
`in_fontable` - current input font table.
`in_lr` - current input font direction.
`direction_table` - table of fonts formatting direction.

Side effects:
None.

23.5 Service Flow Diagram

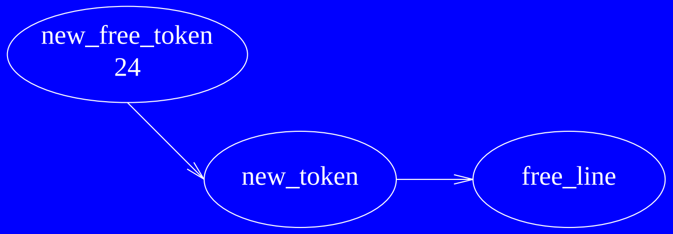


Software Unit #24 — new_free_token

24.1 Software Unit Type

Function group. (lines.c: 34-95 & 350-378)

24.2 Scope Diagram



24.3 Capabilities

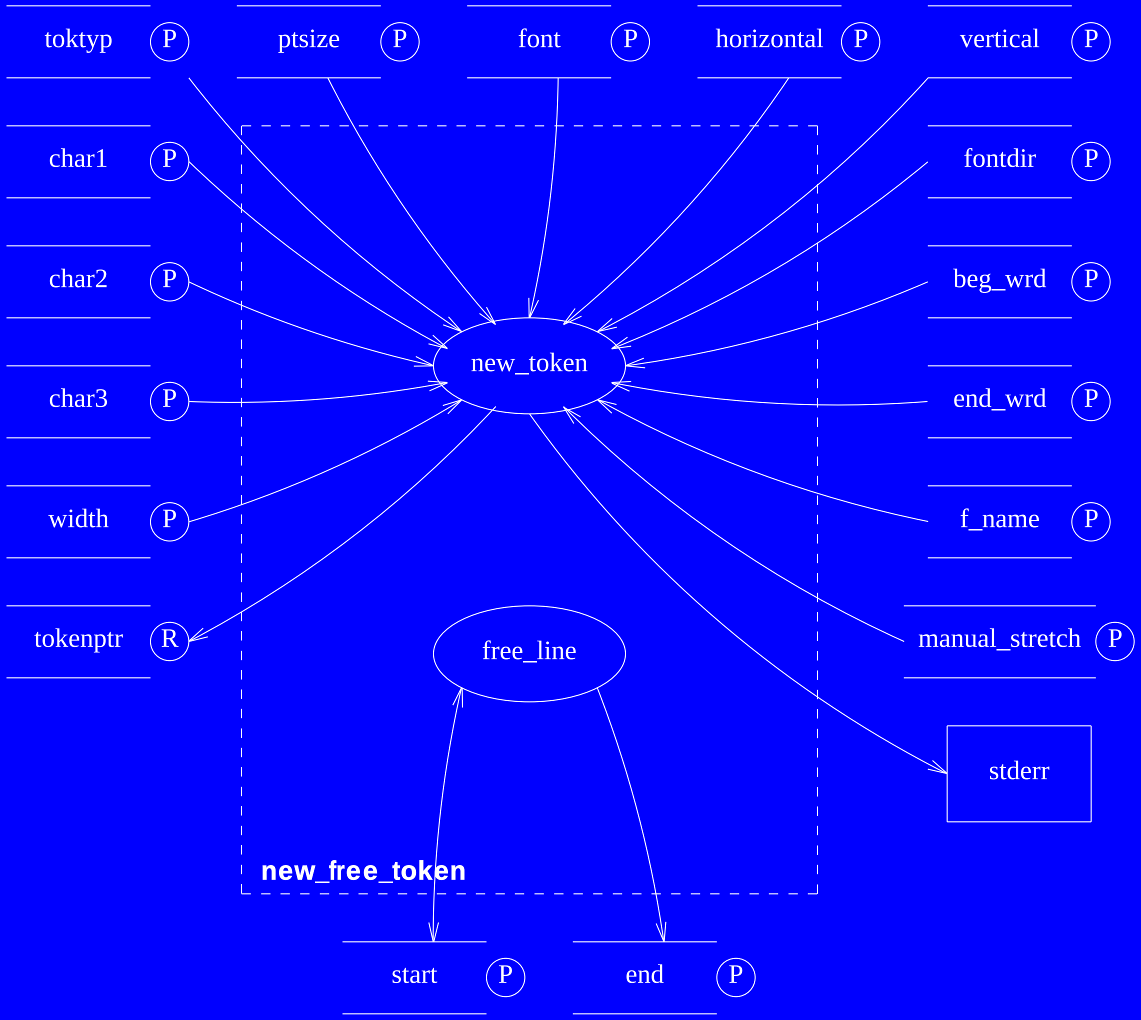
Contains routines to allocate new tokens and to free lines of tokens.

24.4 Interface

Functions:
`new_token` - allocates, initializes and returns a new internal token.
`free_line` - frees the memory allocated to a line of tokens.

Side effects:
1. **`new_token`** allocates memory from the heap.
2. If memory allocation fails in **`new_token`** then an ``out of memory´´ message is printed to stderr and the program halts.
3. **`free_line`** frees allocated memory to the heap.

24.5 Service Flow Diagram

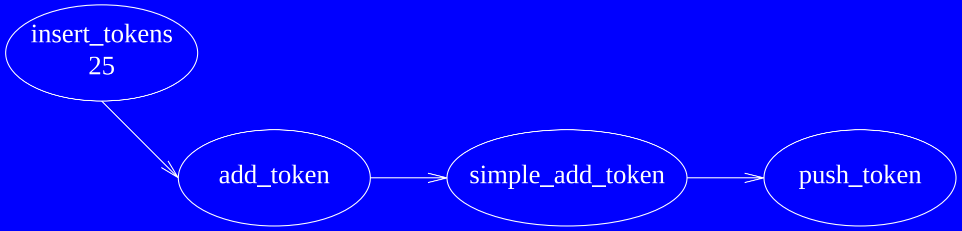


Software Unit #25 — insert_tokens

25.1 Software Unit Type

Procedure group. (lines.c: 141-214)

25.2 Scope Diagram



25.3 Capabilities

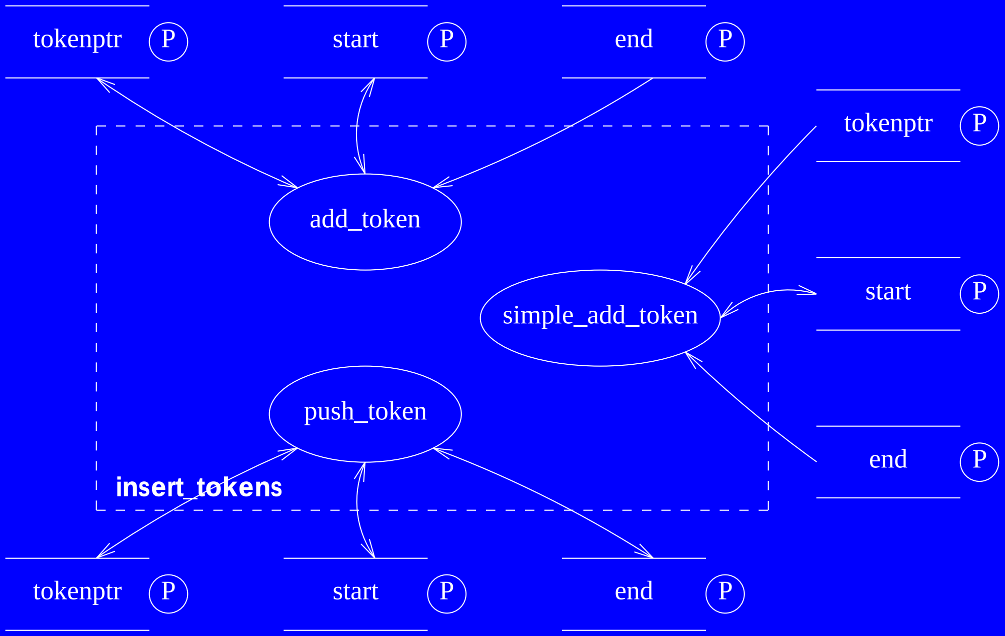
Contains routines to add tokens to the end & front of a token line.

25.4 Interface

Procedures:
`add_token` - adds a token to the end of a line.
`simple_add_token` - adds a token to the end of a line without changing **`tokenptr`**.
`push_token` - pushes a token onto the front of a line.

Side effects:
All procedures change the passed token line.

25.5 Service Flow Diagram

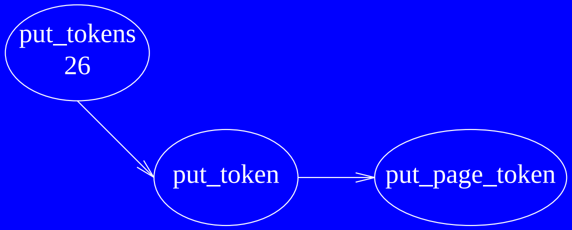


Software Unit #26 — put_tokens

26.1 Software Unit Type

Procedure group. (lines.c: 215-349)

26.2 Scope Diagram



26.3 Capabilities

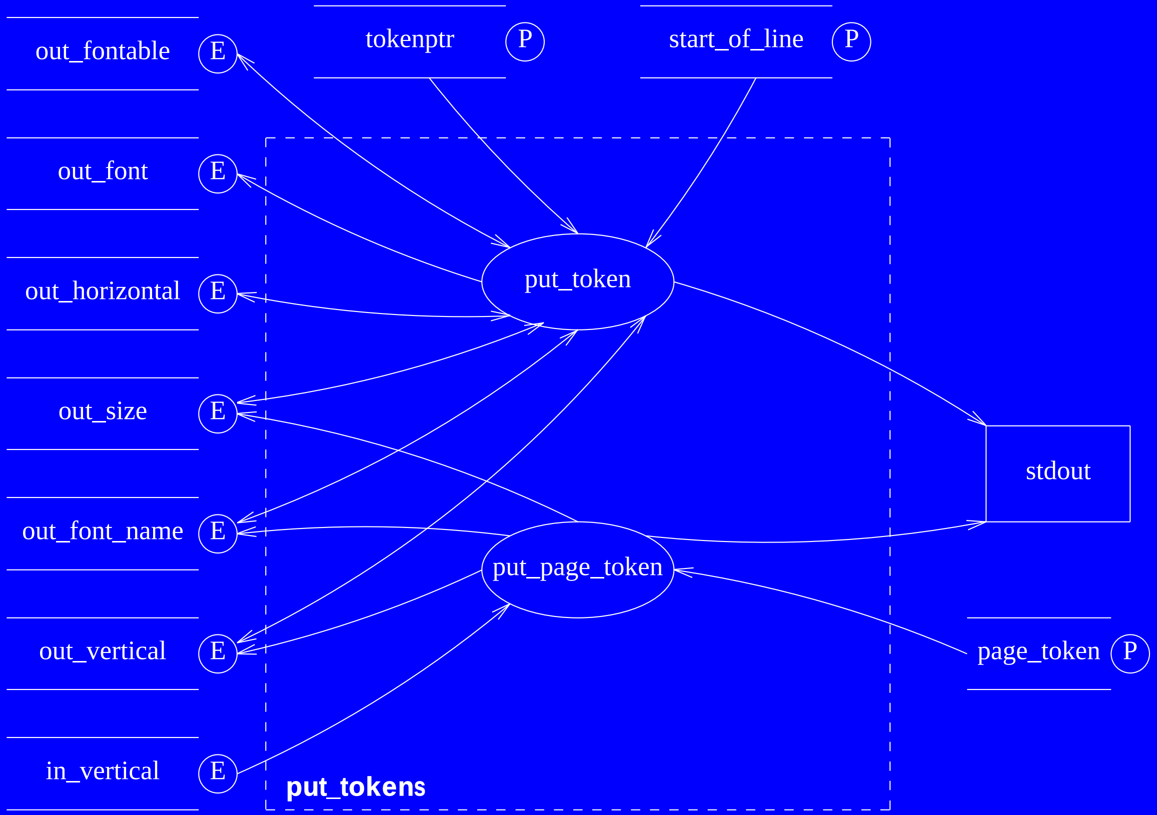
Contains routines to output internal and new page tokens to stdout.

26.4 Interface

Procedures:
put_token - outputs an internal token to stdout.
put_page_token - outputs a new page token and causes next **put_token** call to print font and point sizes.

Side effects:
1. Both procedures print tokens to stdout.
2. **put_token** changes the following external variables: **out_fontable**, **out_font**, **out_horizontal**, **out_size**, **out_font_name**, **out_vertical**.
3. **put_page_token** changes the following external variables: **out_size**, **out_font_name**, **out_vertical**.

26.5 Service Flow Diagram

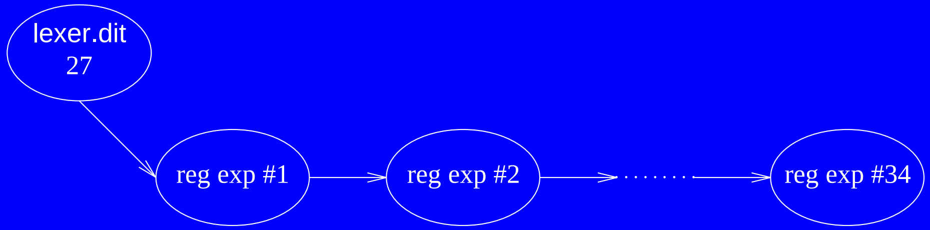


Software Unit #27 — lexer.dit

27.1 Software Unit Type

Lex source file. (lex.dit)

27.2 Scope Diagram



27.3 Capabilities

Contains regular expressions to recognize dtroff output commands, and has associated with each expression an action returning a distinct token.

27.4 Interface

None.

27.5 Service Flow Diagram

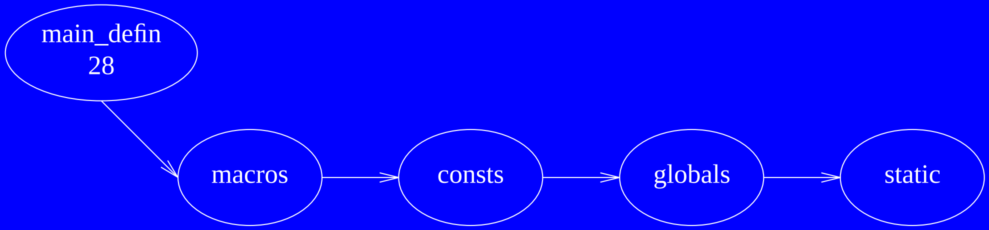
None.

Software Unit #28 — main_defin

28.1 Software Unit Type

Definitions block. (main.c: 1-73)

28.2 Scope Diagram



28.3 Capabilities

Contains definitions used by **`main`** & complete program.

28.4 Interface

Constants:
`USAGE` - command line usage explanation string.

Macros:
`MARK_PREVIOUS_END` - marks the last token in the current input line as ending a word.
`ADD_CHAR1` - creates a new token from 1 char and adds it to end of current input line.
`ADD_CHAR2` - creates a new token from 2 chars and adds it to end of current input line.
`ADD_CHARN` - creates a new token of from 3 chars and adds it to end of current input line.

Static Globals:
`copyright` - string holding copyright information.

Globals:
`in_font` - current input font.
`in_size` - current input point size.
`in_horizontal` - current input horizontal position.
`in_vertical` - current input vertical position.
`in_font_name` - current input font name.
`in_lr` - current input font direction.
`in_fontable` - current input font table.

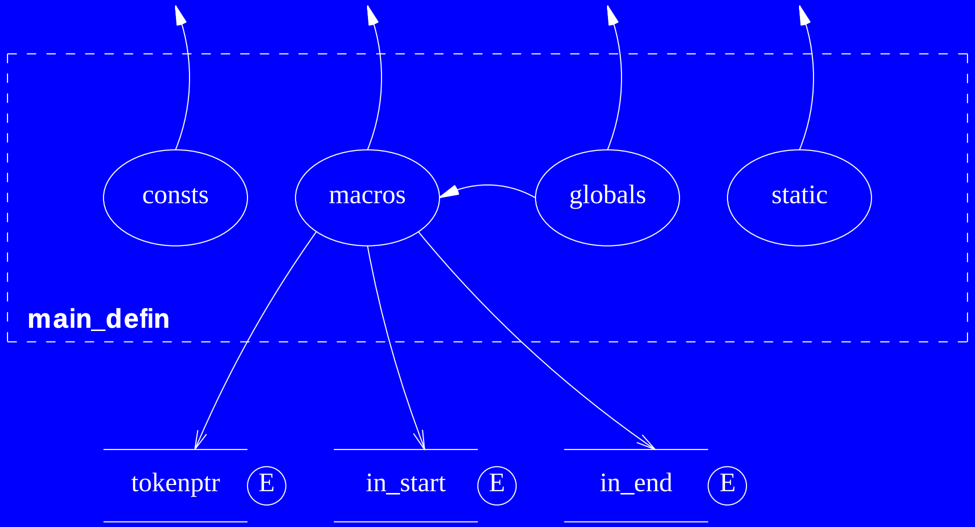
28.4 Interface - Cont

out_font - current output font.
out_size - current output point size.
out_horizontal - current output horizontal position.
out_vertical - current output vertical position.
out_font_name - current output font name.
out_lr - current output font direction.
out_fontable - current output font table.
direction_table - formatting direction of fonts table.
arabic fonts - boolean table stating which font is arabic.
stretch_mode - the stretching mode.
stretch_place - the stretching place.
stretch_amount - the stretch amount in emms.
msc_flag - manually stretched connections control flag.
msl_flag - manually stretched letters control flag.
device - name of output device.
c - general use char for flushing postscript and psfig text.

Side effects:

1. **MARK_PREVIOUS_END** changes the token pointed by **in_end**.
2. **ADD_CHAR1**, **ADD_CHAR2** and **ADD_CHARN** create a new token allocated from the heap and add it to the token line pointed to by **in_start** and **in_end**.

28.5 Service Flow Diagram

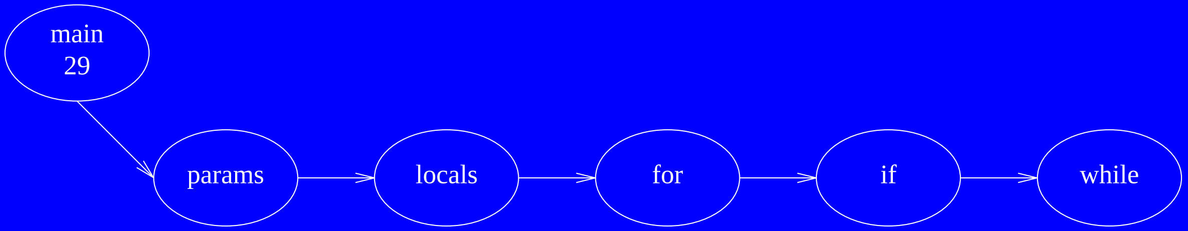


Software Unit #29 — main

29.1 Software Unit Type

Function. (main.c: 74-631)

29.2 Scope Diagram



29.3 Capabilities

Program main function. Initializes the global variables, parses the command line parameters and runs the main ffortid driver routine.

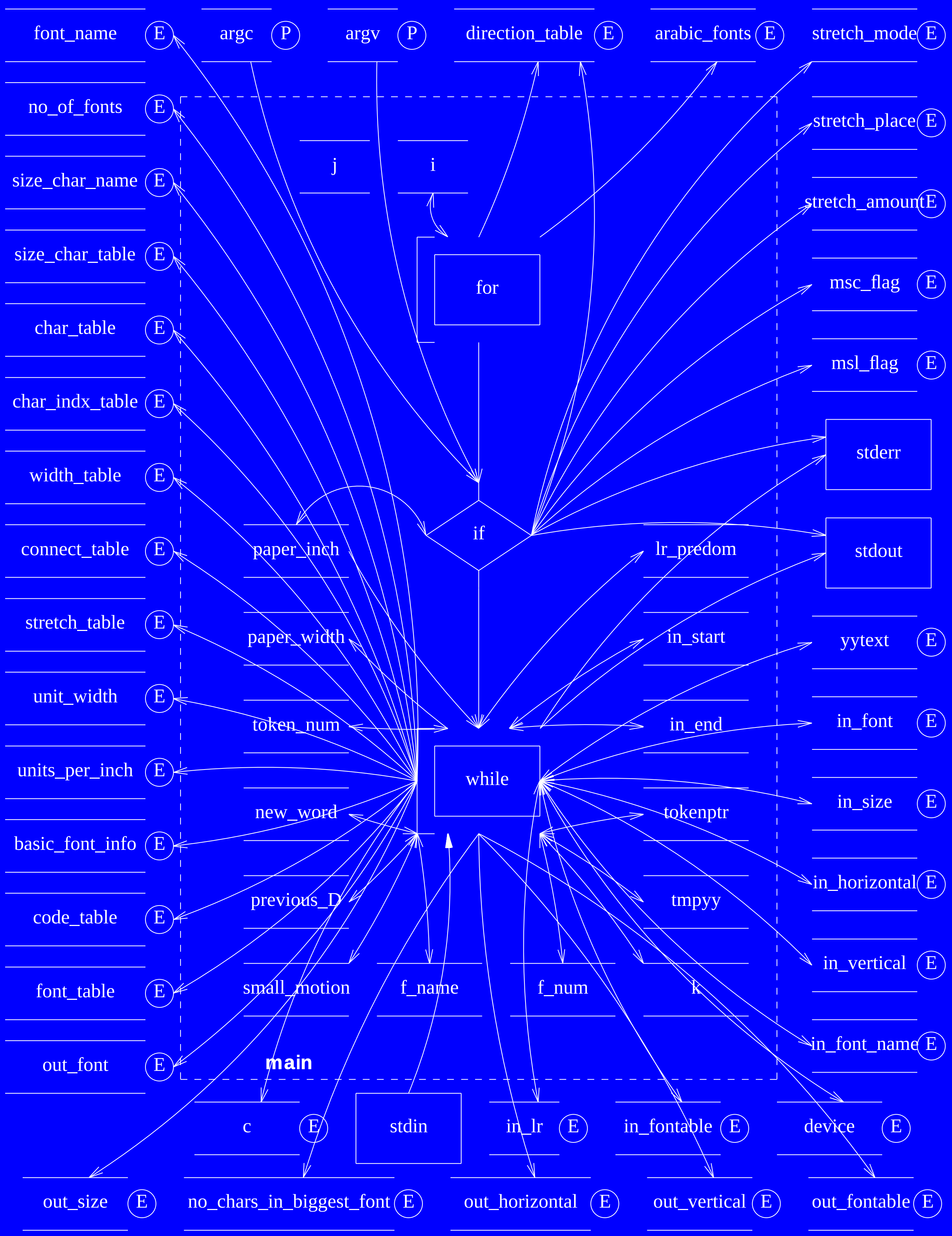
29.4 Interface

Parameters:
`argc` - number of command line arguments.
`argv` - array of pointers to command line arguments.

Return value:
Program exit status.

Side effects:
1. Reads dtroff output from stdin.
2. Prints dtroff output to stdout.
3. Prints encountered errors to stderr and halts program.
4. Allocates and frees memory from the heap.
5. If out of heap memory prints ``out of memory´´ message to stderr and halts program.
6. Changes the following external variables: **`in_font`**, **`in_size`**, **`in_horizontal`**, **`in_vertical`**, **`in_font_name`**, **`in_lr`**, **`in_fontable`**, **`out_font`**, **`out_size`**, **`out_horizontal`**, **`out_vertical`**, **`out_fontable`**, **`direction_table`**, **`arabic_fonts`**, **`stretch_mode`**, **`stretch_place`**, **`stretch_amount`**, **`msc_flag`**, **`msl_flag`**, **`device`**, **`c`**, **`font_name`**, **`no_of_fonts`**, **`size_char_name`**, **`size_char_table`**, **`char_table`**, **`char_indx_table`**, **`width_table`**, **`connect_table`**, **`stretch_table`**, **`unit_width`**, **`units_per_inch`**, **`basic_font_info`**, **`code_table`**, **`font_table`**, **`no_chars_in_biggest_font`**, **`yytext`**.

29.5 Service Flow Diagram

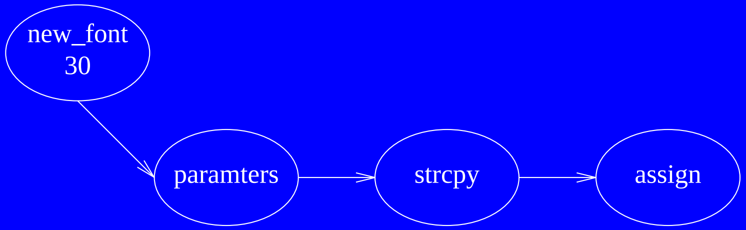


Software Unit #30 — new_font

30.1 Software Unit Type

Procedure. (misc.c: 1-42)

30.2 Scope Diagram



30.3 Capabilities

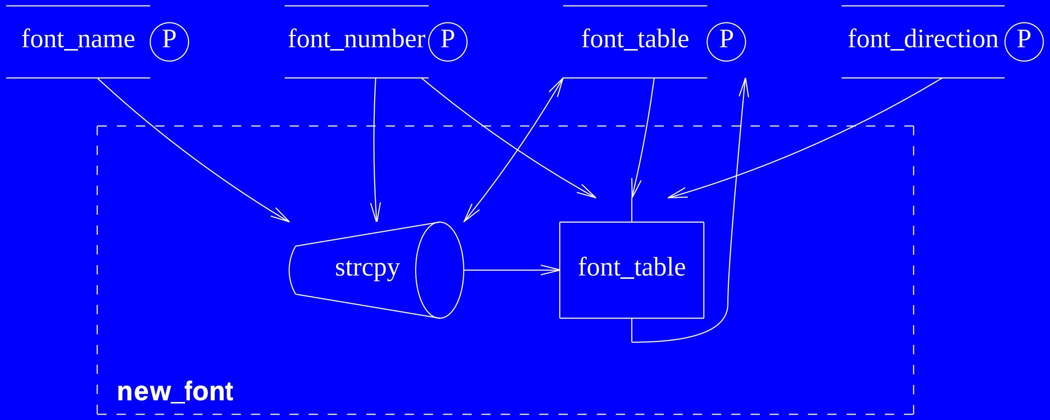
Adds a new font to the font table.

30.4 Interface

Parameters:
`font_number` - number of new font in font table.
`font_name` - string holding font name.
`font_direction` - direction of new font.
`font_table` - the font table to to add the font to.

Side effects:
Changes values in the passed font table.

30.5 Service Flow Diagram

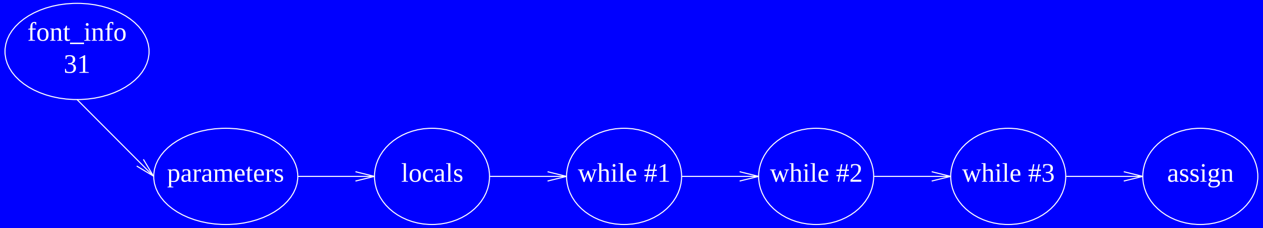


Software Unit #31 — font_info

31.1 Software Unit Type

Procedure. (misc.c: 43-84)

31.2 Scope Diagram



31.3 Capabilities

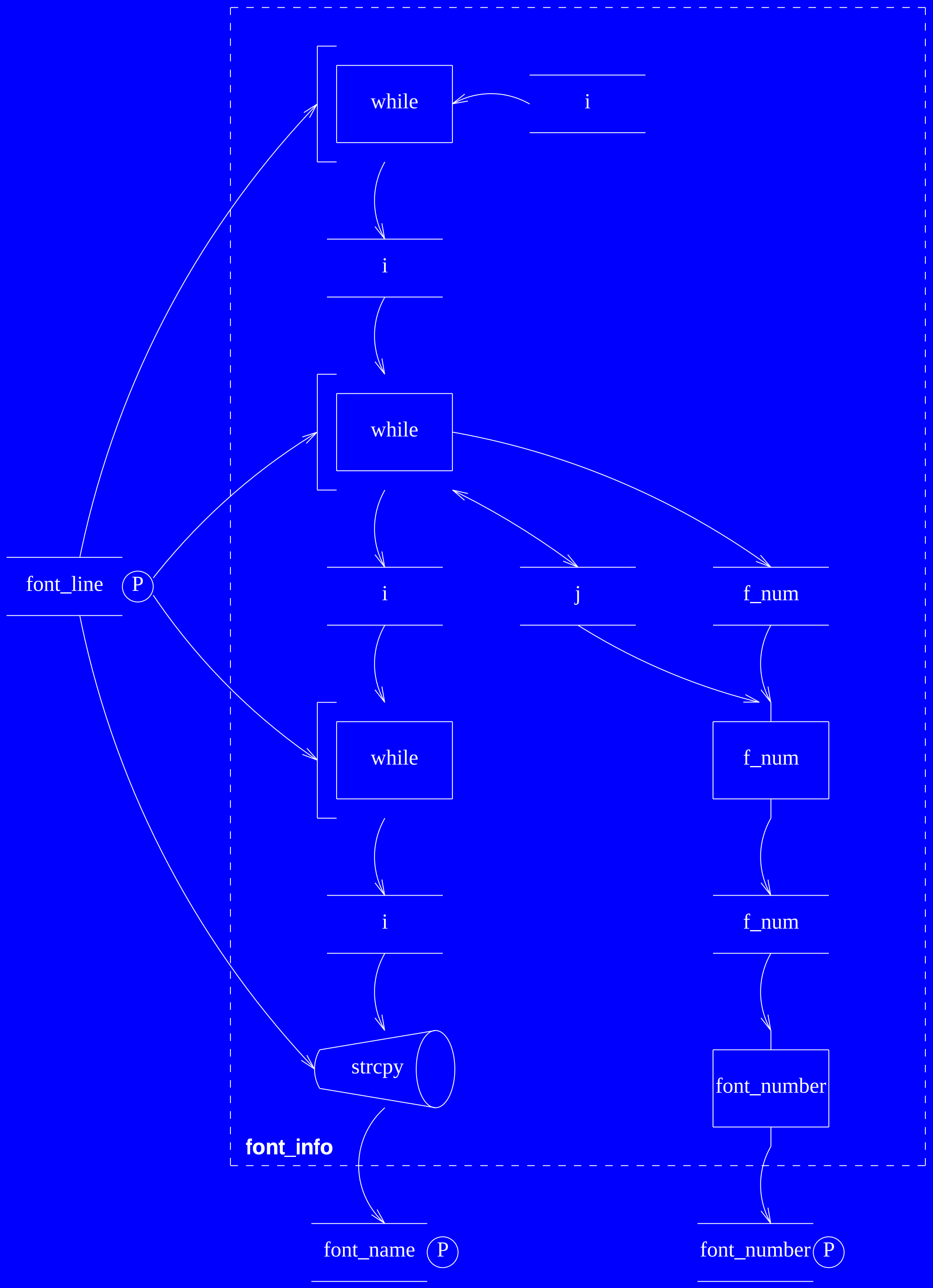
Extracts a font number and name from a font token string.

31.4 Interface

Parameters:
`font_line` - lex font input token line.
`font_number` - pointer to font number.
`font_name` - pointer to font name.

Side effects:
1. Returns through **`font_number`** the font token number.
2. Returns through **`font_name`** the font token name.

31.5 Service Flow Diagram

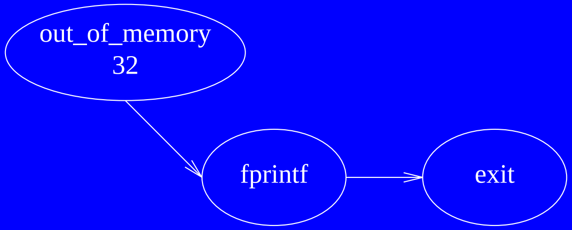


Software Unit #32 — out_of_memory

32.1 Software Unit Type

Procedure. (misc.c: 85-101)

32.2 Scope Diagram



32.3 Capabilities

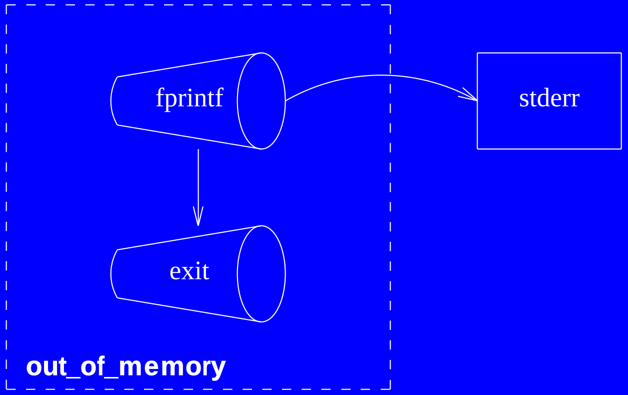
Prints an ``out of memory´´ error message and halts program execution.

32.4 Interface

Parameters:
None.

Side effects:
1. Prints ``out of memory´´ error message to stderr.
2. Causes program to halt.

32.5 Service Flow Diagram

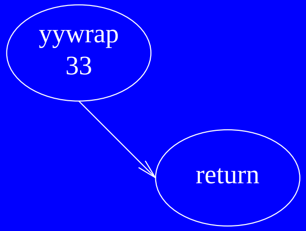


Software Unit #33 — yywrap

33.1 Software Unit Type

Function. (misc.c: 102-114)

33.2 Scope Diagram



33.3 Capabilities

Standard lex library function called whenever lex reaches an end-of-file. The default **`yywrap`** also always returns 1.

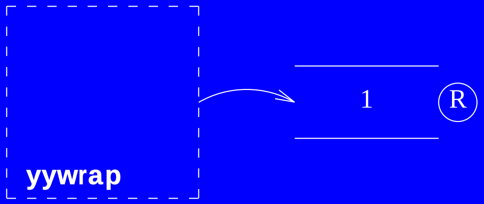
33.4 Interface

Parameters:
None.

Return value:
Always 1.

Side effects:
None.

33.5 Service Flow Diagram

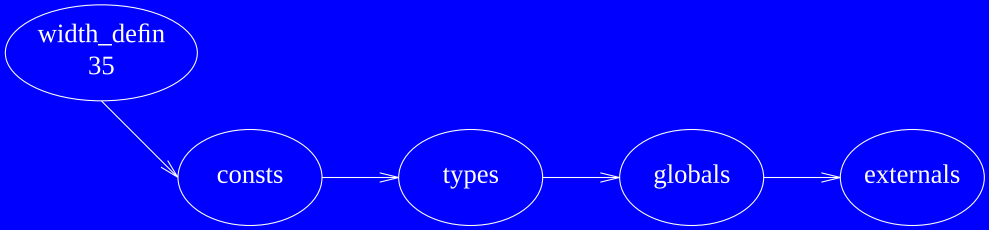


Software Unit #35 — width_defin

35.1 Software Unit Type

Definitions block. (width.c: 1-51)

35.2 Scope Diagram



35.3 Capabilities

Contains definitions and externals used by width.c functions.

35.4 Interface

Constants:
`MAXNOFONTS` - max number of fonts.
`MAXWIDENTRIES` - max width entries.
`NOCHARSINBIGGESTFONT` - no of characters in biggest font in device description.
`MAXNOCHARS` - max number of chars with with two letters or --- names.
`SIZECHARINDXTABLE` - size of character index table including ascii chars but not non-graphics.
`FATAL` - passed to error procedure to signal fatal error.
`BYTEMASK` - mask used to make character numbers positive.

Types:
`Fontinfo` - single font information structure.

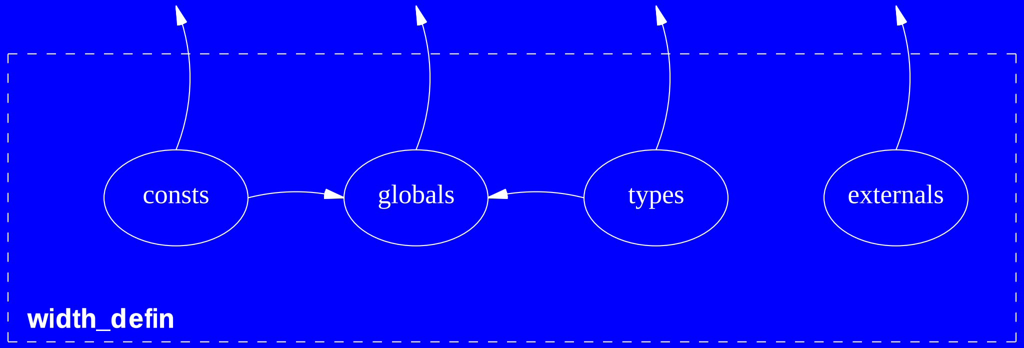
35.4 Interface - Cont

Globals:
`basic_font_info` - array of all fonts information.
`font_name` - array of all font names.
`no_of_fonts` - number of fonts initially mounted on the device.
`indx_1st_spec_font` - index of first special font.
`size_char_table` - size of character table in device.
`unit_width` - basic unit width in device.
`units_per_inch` - number of units per inch in device.
`no_chars_in_biggest_font` - number of chars in biggest font in device.
`size_char_name` - size of character name in device.
`char_name` - array of all character names in device.
`char_table` - array of indexes of characters in char_name.
`char_indx_table` - array of indexes of ascii characters in each font.
`code_table` - array of number codes for each char in each font.
`width_table` - array of widths for each char in each font.
`connect_table` - array of connectivity info for each char in each font.
`stretch_table` - array of stretchability info for each char in each font.
`fontdir` - font files directory.

Externals:
`device` - name of output device.

Side effects:
None

35.5 Service Flow Diagram

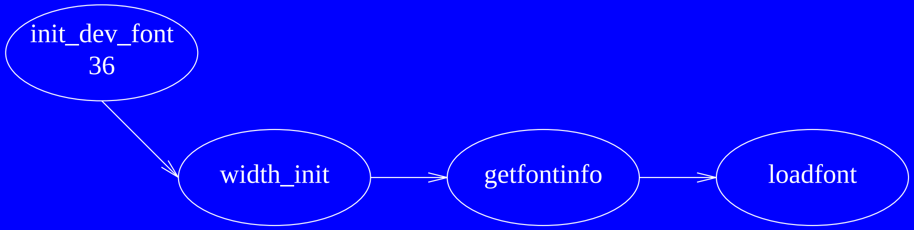


Software Unit #36 — init_dev_font

36.1 Software Unit Type

Procedure group. (width.c: 52-271 & 345-385)

36.2 Scope Diagram



36.3 Capabilities

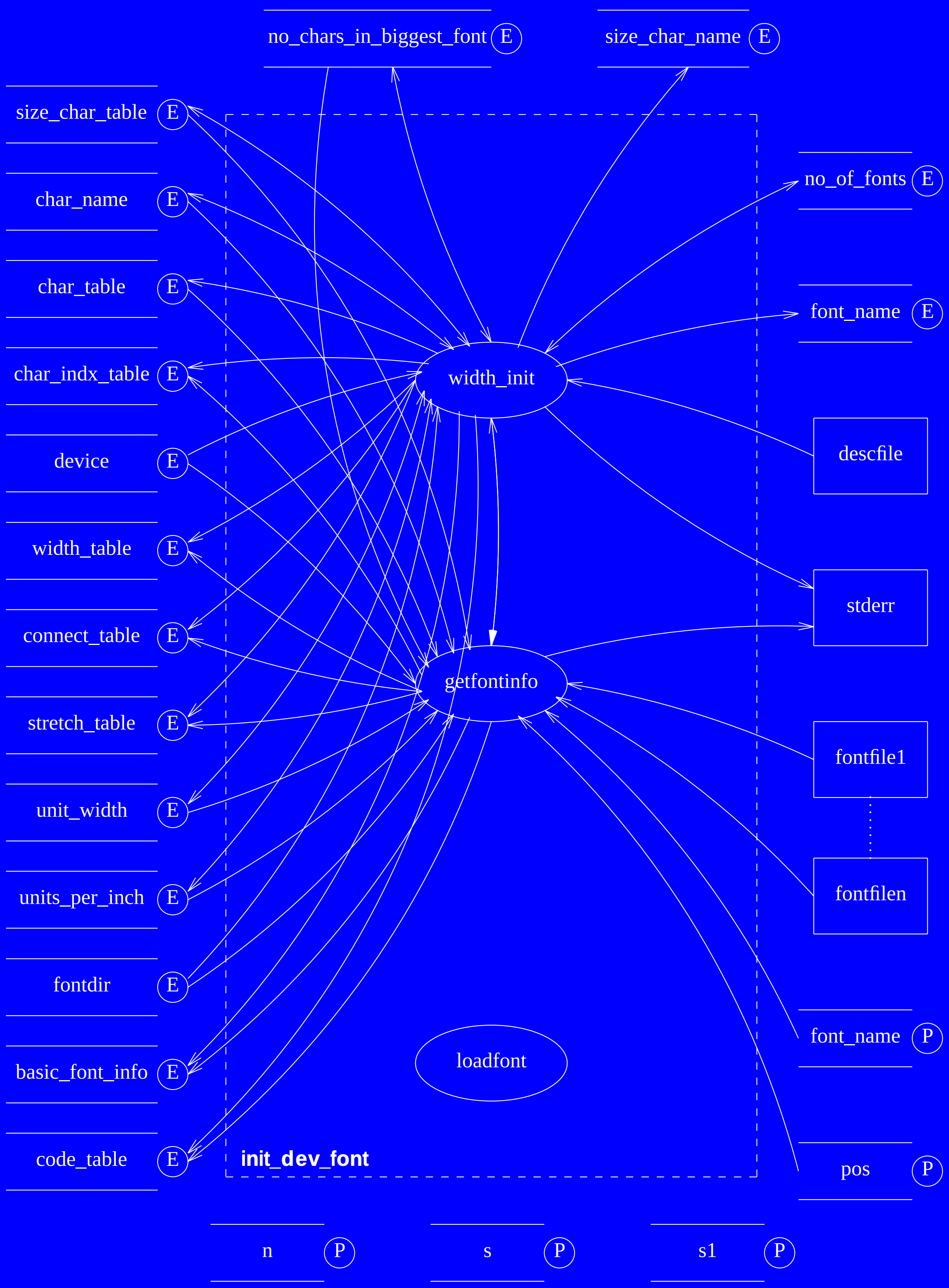
Contains routines to initialize and load device and font width tables.

36.4 Interface

Procedures:
`width_init` - initializes the global device and font tables.
`getfontinfo` - read in a single font table.
`loadfont` - loads a single font table. Currently body commented out.

Side effects:
1. **`width_init`** allocates memory from the heap.
2. Any error found in **`width_init`** is printed to stderr and the program halts.
3. **`width_init`** changes the following external variables: **`size_char_table`**, **`char_name`**, **`char_table`**, **`char_indx_table`**, **`width_table`**, **`connect_table`**, **`stretch_table`**, **`unit_width`**, **`units_per_inch`**, **`basic_font_info`**, **`code_table`**, **`no_chars_in_biggest_font`**, **`size_char_name`**, **`no_of_fonts`**, **`font_name`**.
4. Any error found in **`getfontinfo`** is printed to stderr and the program halts.
5. **`getfontinfo`** changes the following external variables: **`basic_font_info`**, **`width_table`**, **`connect_table`**, **`stretch_table`**, **`code_table`**, **`char_indx_table`**.

36.5 Service Flow Diagram

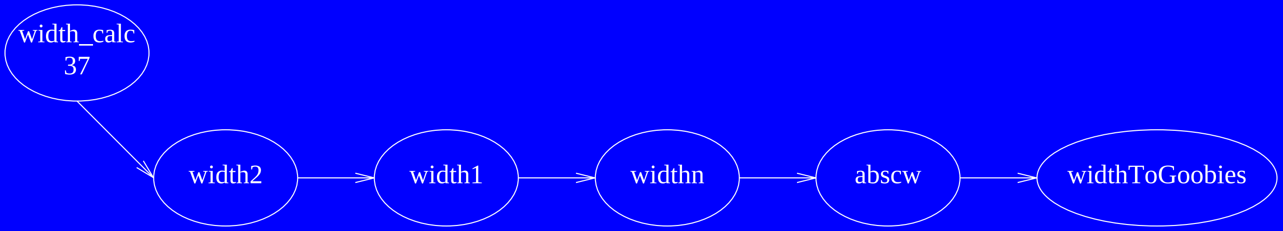


Software Unit #37 — width_calc

37.1 Software Unit Type

Function group. (width.c: 394-544)

37.2 Scope Diagram



37.3 Capabilities

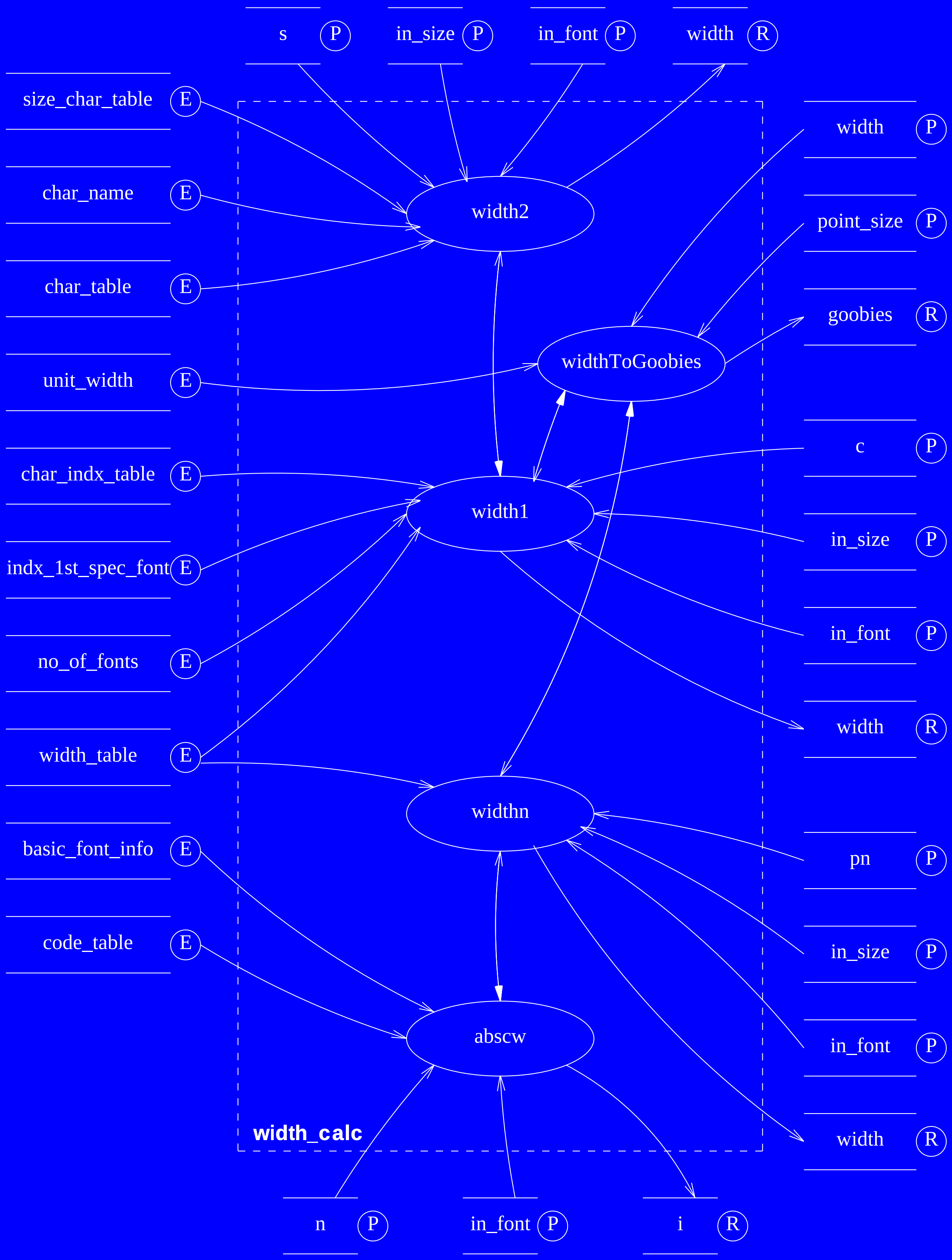
Contains routines to determine the width of different kinds of characters.

37.4 Interface

Functions:
`width2` - returns the width of a specified funny character.
`width1` - returns the width of a specified character.
`widthn` - returns the width of a character specified with its code.
`abscw` - returns the index of char with absolute number **`n`** in font **`in_font`**.
`widthToGoobies` - returns a width at a certain point size in goobies.

Side effects:
None.

37.5 Service Flow Diagram

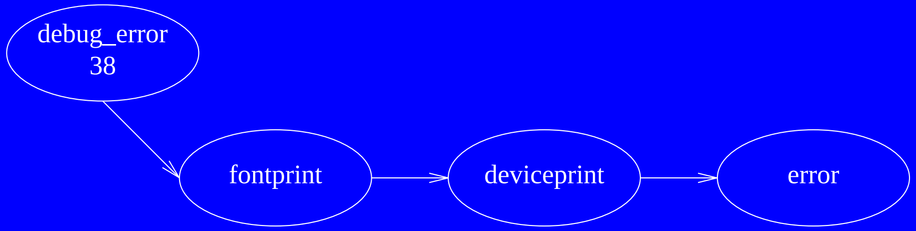


Software Unit #38 — debug_error

38.1 Software Unit Type

Procedure group. (width.c: 272-344 & 385-393)

38.2 Scope Diagram



38.3 Capabilities

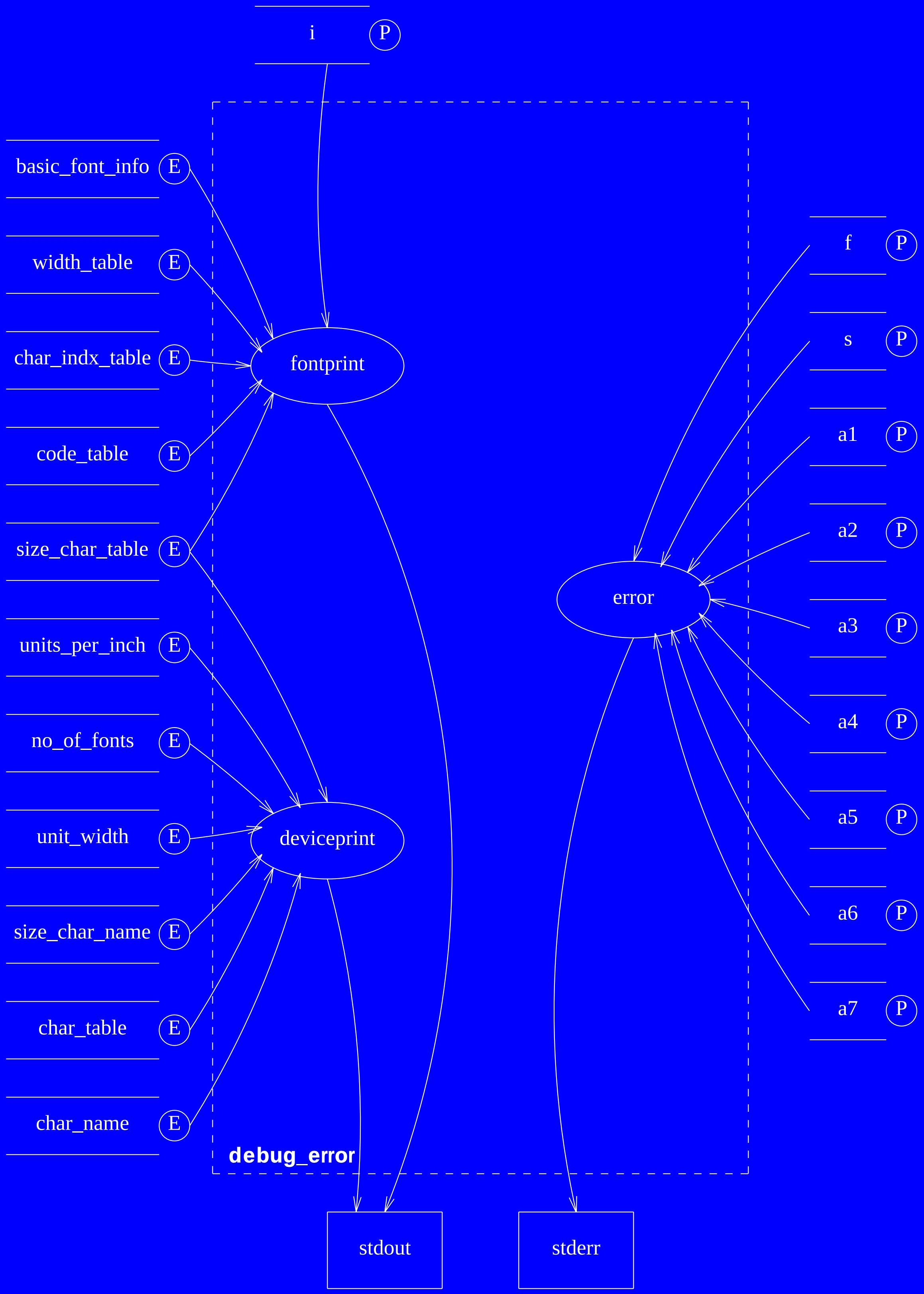
Contains routines that print font width and device tables for debugging and an error routine to print errors and halt program execution if they are fatal.

38.4 Interface

Functions:
`fontprint` - prints a font’s width table.
`deviceprint` - prints the device table.
`error` - prints error message to stderr and halts program if fatal.

Side effects:
1. **`fontprint`** and **`deviceprint`** print to stdout font width & device tables.
2. **`error`** prints to stderr error information
3. **`error`** can halt program execution depending on the **`f`** parameter.

38.5 Service Flow Diagram

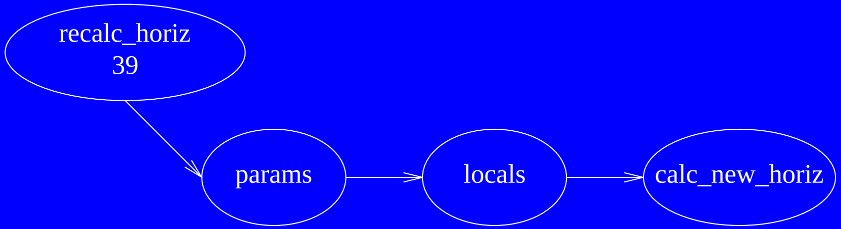


Software Unit #39 — recalc_horiz

39.1 Software Unit Type

Procedure. (dump.c: 243-289)

39.2 Scope Diagram



39.3 Capabilities

Recalculates the horizontal motion in a reversed token line.

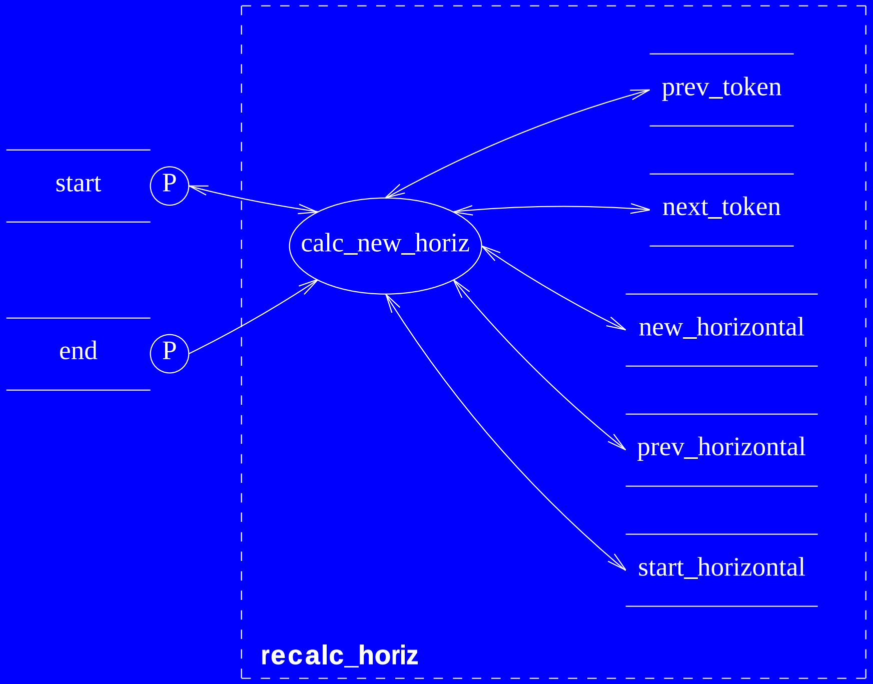
39.4 Interface

Procedure name:
`recalculate_horizontal`

Parameters:
`start` - pointer to first token in the line.
`end` - pointer to last token in the line.

Side effects:
Changes the tokens in the passed token line.

39.5 Service Flow Diagram

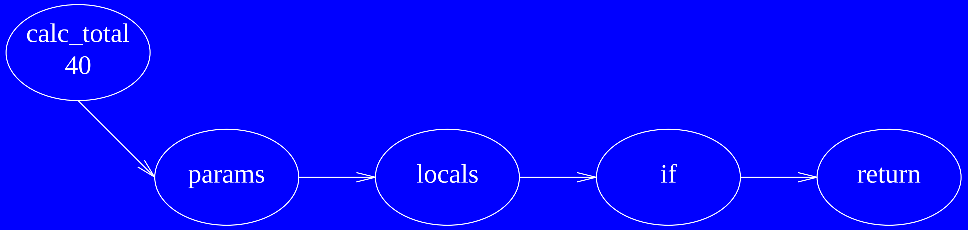


Software Unit #40 — calc_total

40.1 Software Unit Type

Function. (dump.c: 290-344)

40.2 Scope Diagram



40.3 Capabilities

Returns the total stretching amount possible in a line.

40.4 Interface

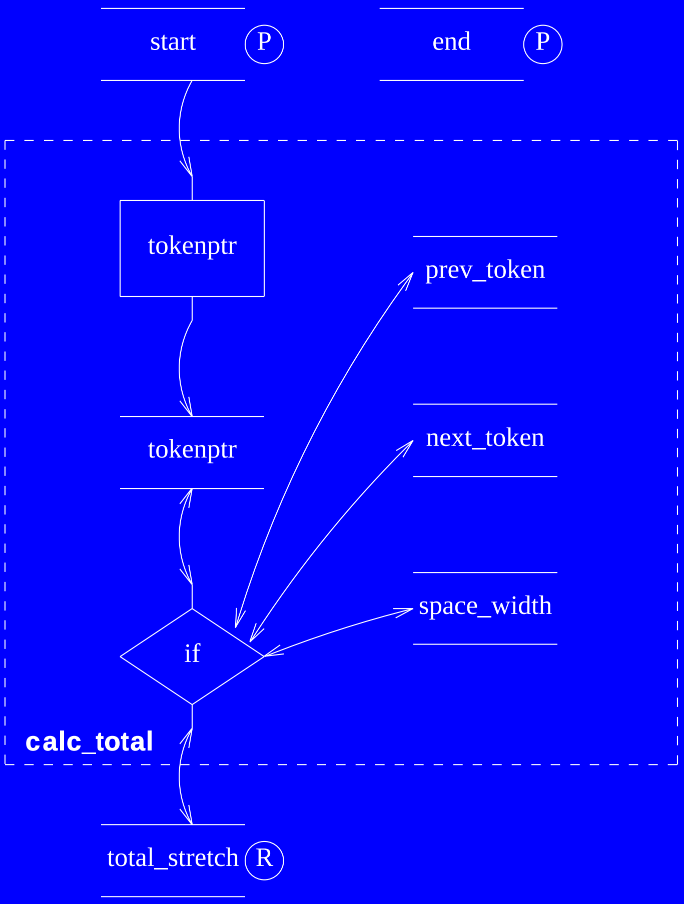
Function name:
`calc_total_stretching`

Parameters:
`start` - pointer to first token in the line.
`end` - pointer to last token in the line.

Return value:
`total_stretch` - total stretch possible in the line.

Side effects:
None.

40.5 Service Flow Diagram

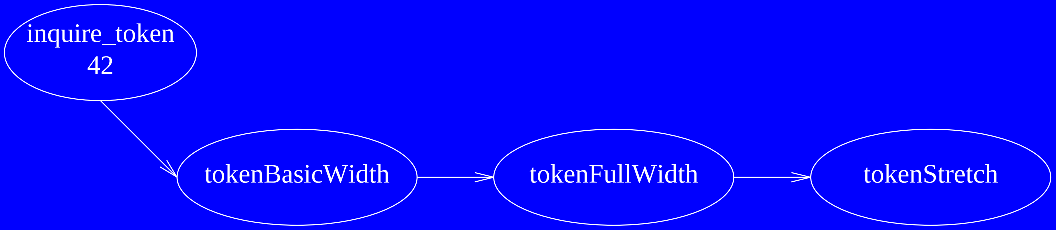


Software Unit #42 — inquire_token

42.1 Software Unit Type

Function group. (lines.c: 96-140)

42.2 Scope Diagram



42.3 Capabilities

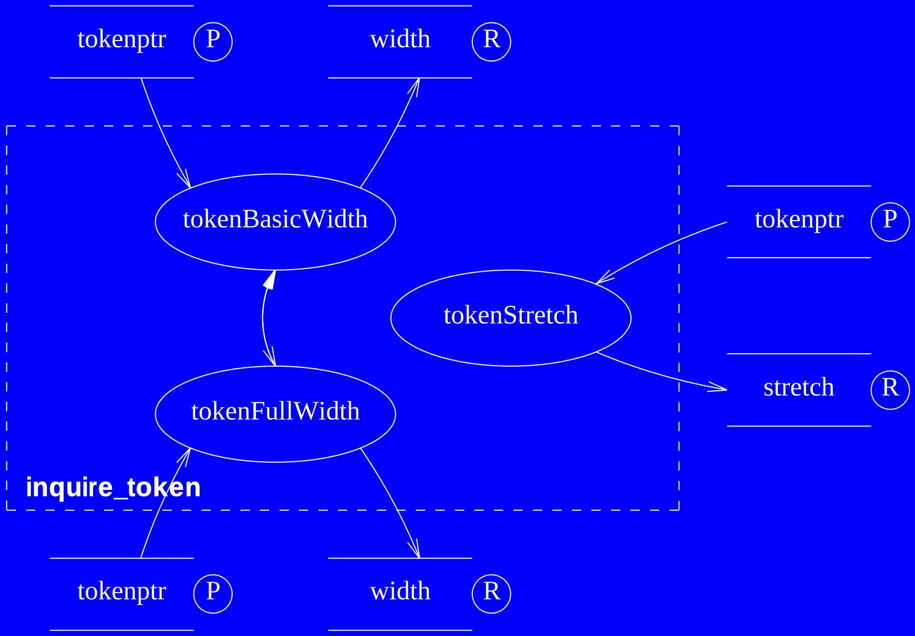
Contains routines to return information about width and stretch of tokens.

42.4 Interface

Functions:
`tokenBasicWidth` - return tokens basic width before stretching.
`tokenFullWidth` - return tokens full width after stretching.
`tokenStretch` - return tokens total stretch amount.

Side effects:
None.

42.5 Service Flow Diagram

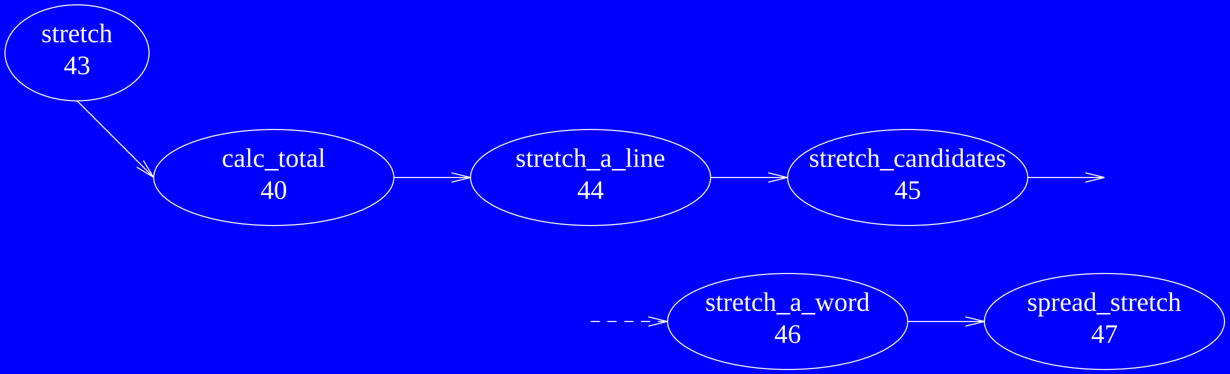


Software Unit #43 — stretch

43.1 Software Unit Type

Function group. (dump.c: 290-896)

43.2 Scope Diagram



43.3 Capabilities

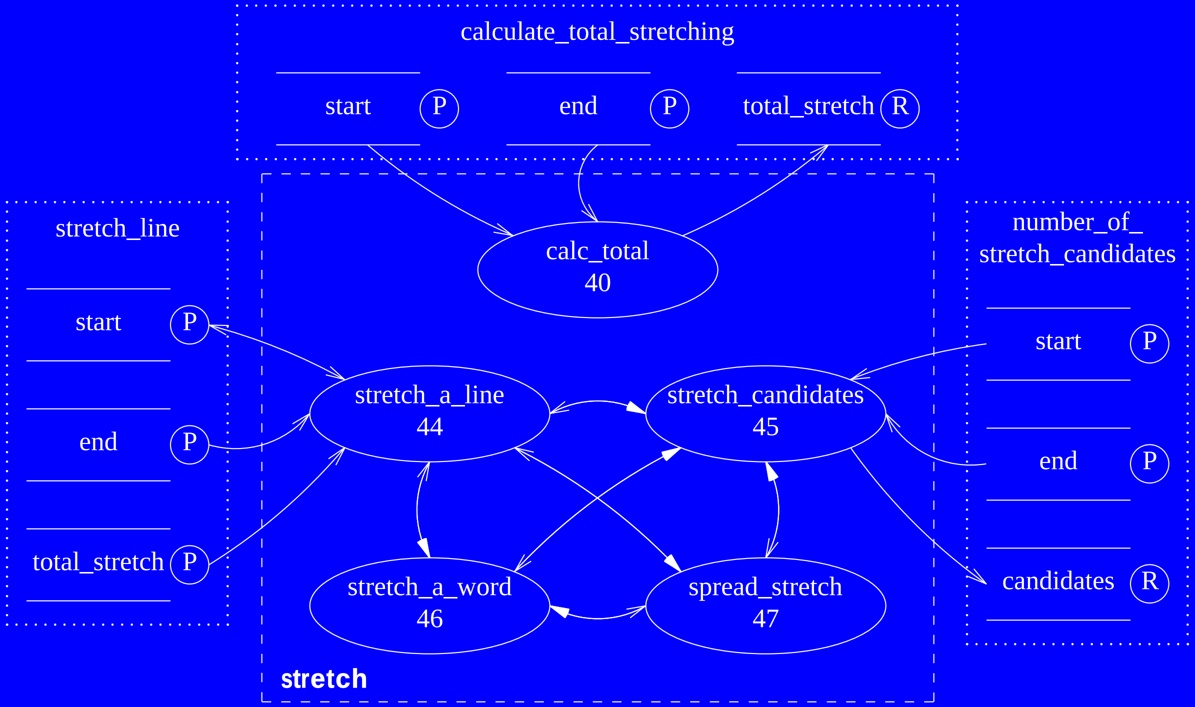
Contains a routine to return the total stretch possible in a line, a routine to stretch a line of tokens according to the current stretch mode and place and a routine to calculate the number of stretch candidates in a line according to the current stretch mode.

43.4 Interface

Functions:
`calc_total_stretching` - returns the total stretch possible in a line.
`stretch_line` - stretches a line by **`total_stretch`** according to the current stretch mode and place.
`number_of_stretch_candidates` - returns the number of stretch candidates in a line according to the current stretch mode.

Side effects:
`stretch_line` changes the passed token line.

43.5 Service Flow Diagram

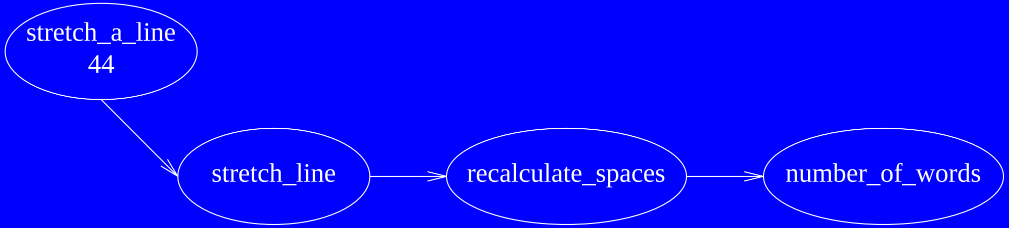


Software Unit #44 — stretch_a_line

44.1 Software Unit Type

Function group. (dump.c: 345-526)

44.2 Scope Diagram



44.3 Capabilities

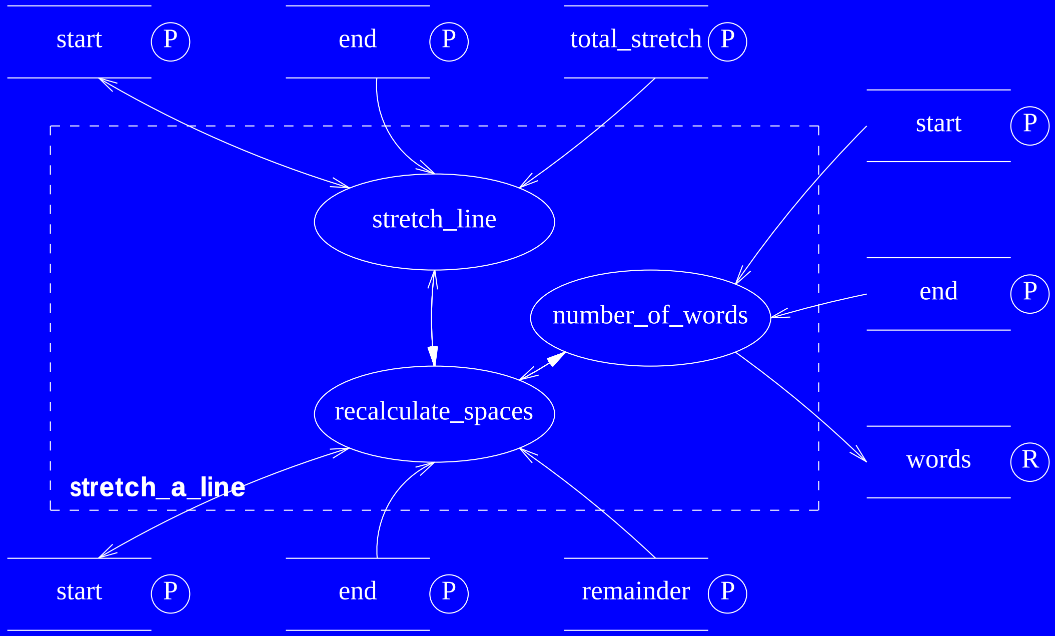
Contains a routine to stretch a line of tokens according to the current stretch mode and place, a routine to recalculate the inter word spaces after a line has been stretched, and a routine to calculate the number of words in a line.

44.4 Interface

Functions:
`stretch_line` - stretches a line by **`total_stretch`** according to the current stretch mode and place.
`recalculate_spaces` - recalculates the inter word spaces in a line and spreads any space remainder after the line has been stretched.
`number_of_words` - returns the number of words in a line.

Side effects:
`stretch_line` and **`recalculate_spaces`** change the passed token line.

44.5 Service Flow Diagram

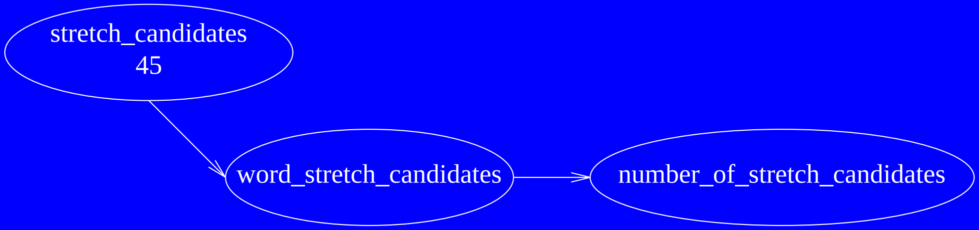


Software Unit #45 — stretch_candidates

45.1 Software Unit Type

Function group. (dump.c: 527-657)

45.2 Scope Diagram



45.3 Capabilities

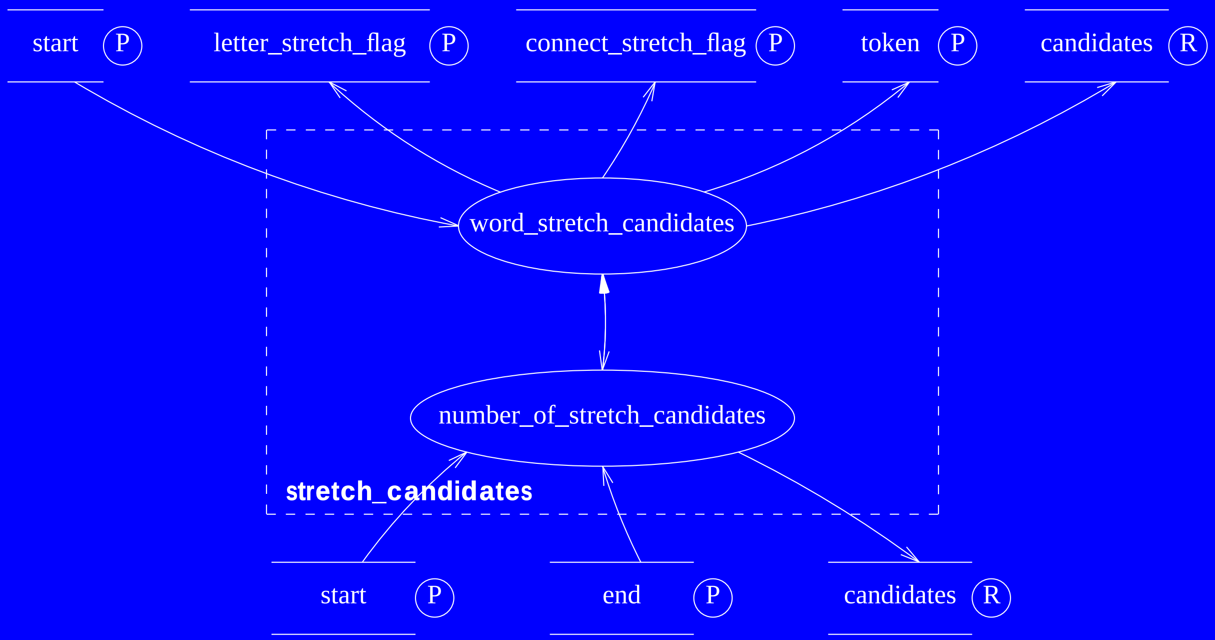
Contains routines to return the number of stretch candidates in a word or line according to the current stretch mode.

45.4 Interface

Functions:
`word_stretch_candidates` - returns the number of stretch candidates in a word according to the current stretch mode and a pointer to the stretch token candidate including flags indicating if the letter or connection (or both) are the candidates.
`number_of_stretch_candidates` - returns the number of stretch candidates in a line according to the current stretch mode.

Side effects:
None.

45.5 Service Flow Diagram

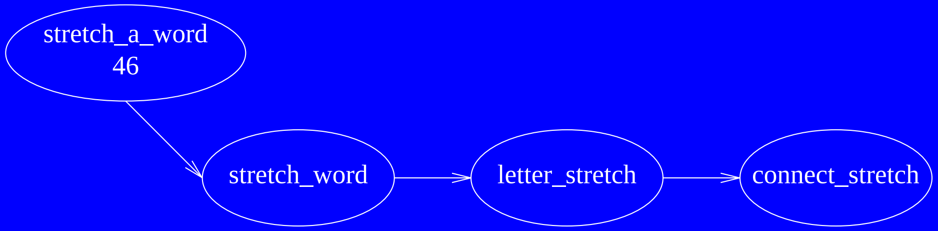


Software Unit #46 — stretch_a_word

46.1 Software Unit Type

Function group. (dump.c: 658-773)

46.2 Scope Diagram



46.3 Capabilities

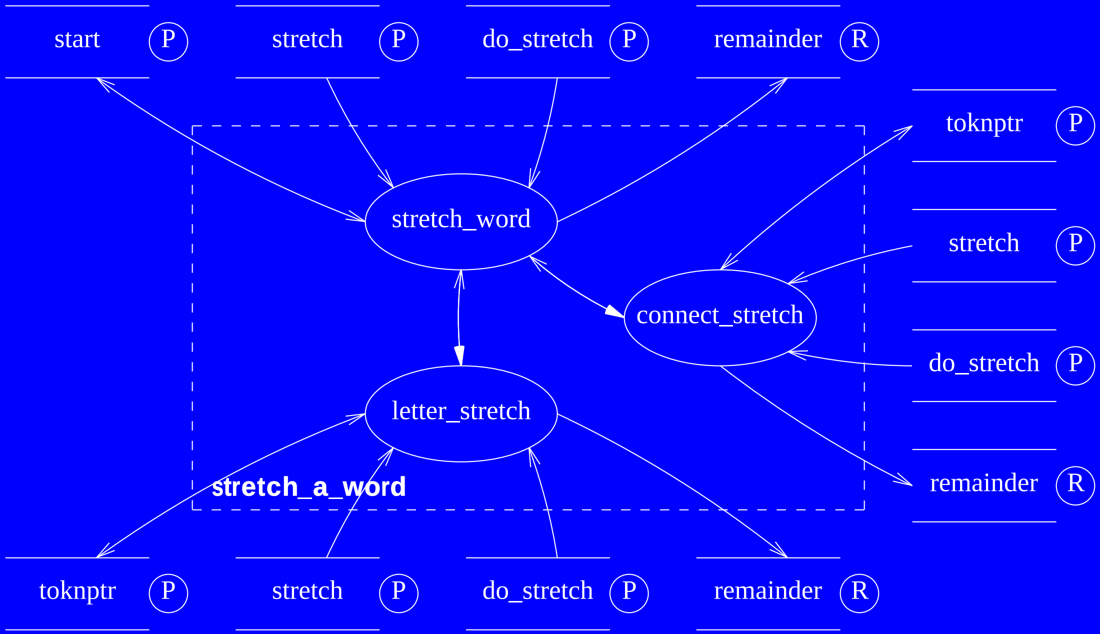
Contains functions to stretch a word of tokens according to the current stretch mode and place and functions to stretch a letter or connection of a single token.

46.4 Interface

Functions:
`stretch_word` - stretches all the stretch units of a word by **`stretch`** according to the current stretch mode and place and returns the remainder. If the parameter **`do_stretch`** is false then no actual stretching is preformed.
`letter_stretch` - stretches a single letter by **`stretch`** and returns the remainder. If the parameter **`do_stretch`** is false then no actual stretching is preformed.
`connect_stretch` - stretches a single connection to a letter by **`stretch`** and returns the remainder. If the paremeter **`do_stretch`** is false then no actual stretching is preformed.

Side effects:
`stretch_word`, **`letter_stretch`** and **`connect_stretch`** can change the passed tokens.

46.5 Service Flow Diagram

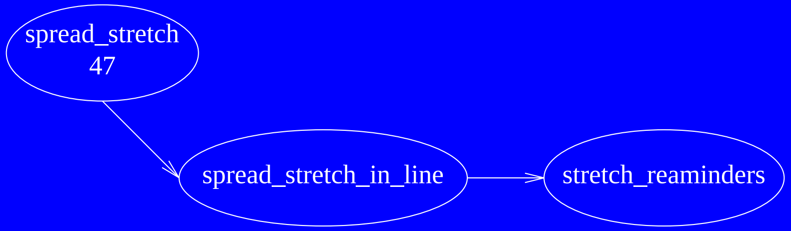


Software Unit #47 — spread_stretch

47.1 Software Unit Type

Function group. (dump.c: 774-896)

47.2 Scope Diagram



47.3 Capabilities

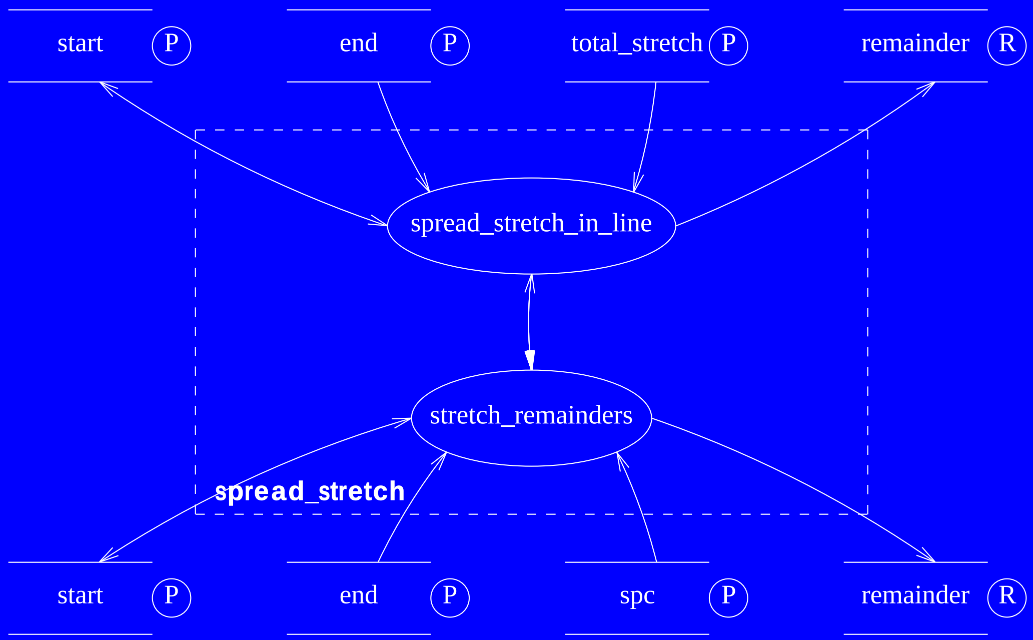
Contains a function to stretch all the stretchable units in a line according to the stretch mode and place by dividing the total stretch evenly among them, and a function to calculate the total remainder from a stretch of all the stretchable units in a line by a certain stretch amount.

47.4 Interface

Functions:
`spread_stretch_in_line` - stretches all the stretchable units in a line according to the stretch mode and place by dividing the total stretch evenly among them, and returns the remainder, if any.
`stretch_remainders` - returns the total remainders from all the stretchable units in a line assuming they are stretched with stretch **`spc`**.

Side effects:
`spread_stretch_in_line` changes the tokens in the passed token line.

47.5 Service Flow Diagram

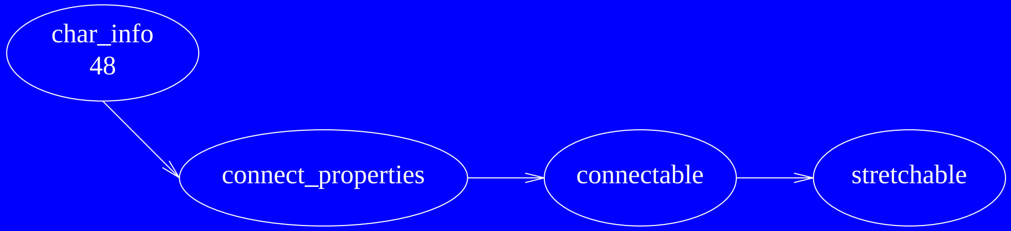


Software Unit #48 — char_info

48.1 Software Unit Type

Function group. (width.c: 545-596)

48.2 Scope Diagram



48.3 Capabilities

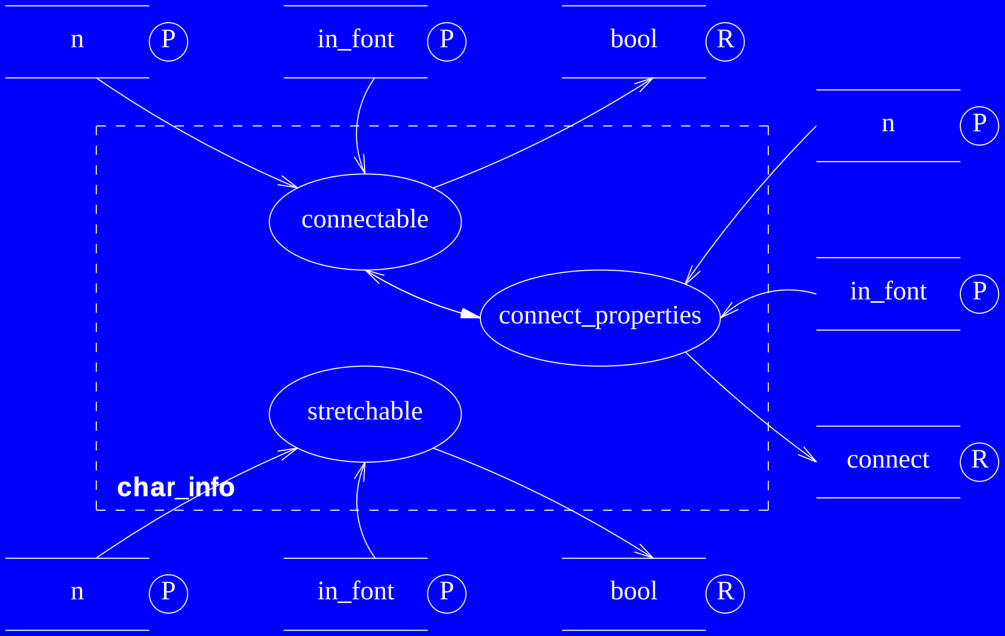
Contains routines to return connectability and stretchability of chars.

48.4 Interface

Functions:
`connect_properties` - returns the connectivity of absolute char **`n`**.
`connectable` - returns whether absolute char **`n`** is a connect previous letter.
`stretchable` - returns whether absolute char **`n`** is stretchable.

Side effects:
None.

48.5 Service Flow Diagram



4. Acknowledgments

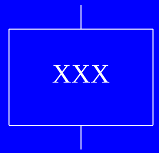
I wish to thank Prof. Daniel Berry for his helpful comments and assistance in preparing this manual. I would also like to thank Prof. Noah Prywes for his guidance and advice.

Appendix A - Service Flow Diagram Icons

Appendix B - ffortid Manual

Appendix C - ffortid Source Files

Appendix A - Service Flow Diagram Icons

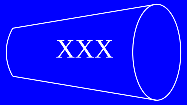


Assignment

a=b+c

XXX is the name of the variable on the left hand side of the assignment.

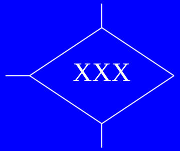
Procedure/Function Call



my_procedure(arg1,arg2)

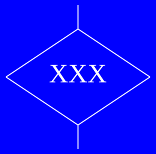
XXX is the procedure name.

Condition



if (my_var)...else... *switch(c)*

XXX is either IF or SWITCH.

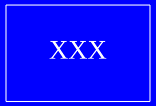


Simple Condition

if (my_var)

XXX is always IF without an else.

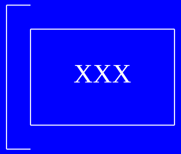
IO File



FILE fd;*

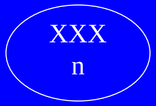
XXX is the name of the variable or the name of the file in quotes.

Loop



for(i=0;i<n;i++)... while (cond) do ... do statement while (cond)

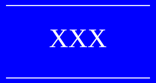
XXX is the type of statment, e.g. FOR, WHILE or DO.



Software Unit

A single Software Unit.

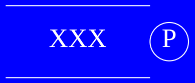
XXX is the name of the SWU. If it has a number the number n is shown.



Local Variable

type name;

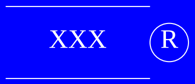
XXX is the name of the variable.



Parameter Variable

func(type name);

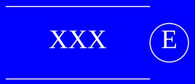
XXX is the name of the parameter variable.



Return Variable

type func(arg1, arg2);

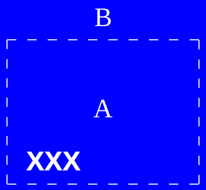
XXX is the name of the return variable.



External Variable

extern type name;

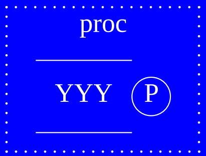
XXX is the name of the external var;



Software Unit Borderline

A is internal to SWU XXX. B is external.

XXX is the name of the SWU. All SWU in the scope of XXX are in the box.



Function/Procedure parameters group

proc is the name of the function/procedure. YYY is a parameter.

Groups function/procedure parameters and return value for SWU entry point.

$$\begin{tikzcd} A \arrow[r] & B \end{tikzcd}$$

Scope relationship

SWU A precedes SWU B in a block.

Captures precedence of SWU within a block.

$$\begin{tikzcd} A \arrow[r, bend left] & B \end{tikzcd}$$

Data Flow Relationship

Data flows between SWU A and SWU B.

Relationship between a consumer and producer of data.

$$\begin{tikzcd} A \arrow[r, bend left, leftrightarrow] & B \end{tikzcd}$$

Bi-Directional Data Flow Relationship

Data flows between SWU A and SWU B and vice-versa.

Bi-Directional relationship between a consumer and producer of data.

Call Relationship

$$\begin{tikzcd} A \arrow[r, bend left, leftrightarrow] & B \end{tikzcd}$$

SWU A calls a function or procedure in SWU B.

Relationship between a function/procedure caller and the callee.

Use relationship

$$\begin{tikzcd} A \arrow[r, bend left] & B \end{tikzcd}$$

SWU B uses declerations or definitions in SWU A.

Relationship between decleration/definition in a SWU and it’s use in another SWU.

Appendix B - ffortid Manual Page

NAME

ffortid – in dtroff output, find and reverse all text in designated right-to-left fonts and carry out stretching in Arabic, Hebrew, and Persian text

SYNOPSIS

ffortid [–**r***font-position-list*] [–**w***paperwidth*] [–**a***font-position-list*] ...
[–**s**[**n**|[[**l**|**c**|**e**|**b**][**f**|**2**|**m**[*amount*]|**a**|**ad**|**al**]]] [–**ms**[**c**|**l**] ...

DESCRIPTION

ffortid's job is to take the *dtroff*(1) output which is formatted strictly left-to-right, to find occurrences of text in a right-to-left font and to rearrange each line so that the text in each font is written in its proper direction. *ffortid* deals exclusively with *dtroff* output, it does not know and does not need to know anything about any of *dtroff*'s preprocessors. Therefore, the results of using *ffortid* with any of *dtroff*'s preprocessors depends only on the *dtroff* output generated as a result of the input to *dtroff* from the preprocessors. Furthermore, the output of *ffortid* goes on to the same device drivers to which the *dtroff* output would go; therefore, *ffortid*'s output is in the same form as that of *dtroff*. *ffortid* reads its input from the standard input and write to the standard output.

In the command line, the –**r***font-position-list* argument is used to specify which font positions are to be considered right-to-left. A *font-position-list* is a list of font positions separated by white space, but with no white space at the beginning. *ffortid*, like *ditroff*, recognizes up to 256 possible font positions (0-255). The actual number of available font positions depends only on the typesetting device and its associated *dtroff* device driver. The default font direction for all possible font positions is left-to-right. Once the font direction is set, either by default or with the –**r***font-position-list* argument, it remains in effect throughout the entire document. Observe then that *ffortid*'s processing is independent of what glyphs actually get printed for the mounted fonts. It processes the designated fonts as right-to-left fonts even if, in fact, the alphabet is that of a left-to-right language. In fact, it is possible that the same font be mounted in two different positions, only one of which is designated as a right-to-left font position. This is how a single font can be printed left-to-right and right-to-left in the same document.

In addition to the specified font directions, the results of *ffortid*'s reformatting also depends on the document's *current formatting direction*, which can be either left-to-right or right-to-left. The default formatting direction is left-to-right and can be changed by the user at any point in the document through the use of the

```
x X PL
```

and

```
x X PR
```

commands in the *dtroff* output. These commands set the current formatting direction to left-to-right and right-to-left, respectively. These commands are either meaningless or erroneous to *dtroff* device drivers; therefore they are removed by *ffortid* as they are obeyed. These commands can be generated by use of

```
\X'PL'
```

and

```
\X'PR'
```

escapes in the *dtroff* input. For the convenience of the user, two macros

```
.PL
```

and

```
.PR
```

are defined in the **−mX2** and **−mXP** macro packages, that cause generation of the proper input to *ffortid*. They are defined by

```
..de PL
\\X'PL'
..
.de PR
\\X'PR'
..
```

If the current formatting direction is left-to-right, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from left to right. In each *dtroff* output line, any sequence of contiguous right-to-left font characters is reversed in place.

If the current formatting direction is right-to-left, all formatting, filling, adjusting, indenting, etc. is to appear as occurring from right to left. Each *dtroff* output line is reversed, including both the left and right margins. Then, any sequence of contiguous left-to-right font characters is reversed in place.

The −*`wpaperwidth`* argument is used to specify the width of the paper, in inches, on which the document will be printed. As explained later, *ffortid* uses the specified paper width to determine the width of the right margin. The default paper width is 8.5 inches and like the font directions, it remains in effect throughout the entire document.

It is important to note that *ffortid* uses the specified paper width to determine the margin widths in the reformated output line. For instance, suppose that a document is formatted for printing on paper 8.5 inches wide with a left margin (page offset) of 1.5 inches and a line length of 6 inches. This results in a right margin of 1 inch. Suppose then that the text specifies a current formatting direction of right-to-left. Then, *ffortid*'s reformatting of the output line results in left and right margins of 1 and 1.5 inches, respectively. This margin calculation works well for documents formatted entirely in one direction. However, as a document's formatting direction changes, the resulting margins widths are exchanged. Thus, **`.PL`**'s right and left margins end up not being the same as **`.PR`**'s right and left margins. The user can make *ffortid* preserve the left and right margins by specifying, with the −*`wpaperwidth`* argument, a paper width other than the actual paper width. This artificial paper width should be chosen so that both margins will appear to *ffortid* to be as wide as the desired left margin. For example, for the document mentioned above, a specified paper width of 9.0 inches results in reformatted left and right margins of 1.5 inches each. The resulting excess in the right margin is just white space that effectively falls of the edge of the paper and does not effect the formatting of the document.

There is one exception to these simple rotation rules in that *ffortid*, at present, makes no attempt to reverse any of *dtroff*'s drawing functions, such as those used by *pic*(1) and *ideal*(1) (which are also available directly to the user). It is therefore suggested that these drawing functions, and thus *pic* and *ideal*, be used only when the current formatting direction is left-to-right. Additionally, due to the cleverness of the *dtroff* output generated by most substantial *eqn*(1) equations, it is suggested that *eqn*'s use also be limited to a left-to-right formatting direction for all but the simplest forms of equations. These rules do not in any way restrict the use of right-to-left fonts in the text dealt with by any of the preprocessors, but simply suggest that these particular preprocessors be used only when the formatting direction is left-to-right.

An additional point to keep in mind when preparing input both for *dtroff*'s preprocessors and for *dtroff* itself is that *ffortid* rotates, as a unit, each sequence of characters of the same direction. In order to force *ffortid* to rotate parts of a sequence independently, as for a *tbl*(1) table, one must artificially separate them with a change to a font of the opposite direction.

The −**a***font-position-list* argument is used to indicate which fonts positions, generally a subset of those designated as right-to-left, contain fonts for Arabic, Hebrew, Persian, or related languages, whose fonts support stretching of letters and/or connections. For these fonts, left and right justification of a line can be achieved by stretching instead of inserting extra white space between the words in the line. If requested by use of the −**s** argument described below, stretching is done on a line only if the line contains at least one word in a −**a** designated font. If so, stretching is used in place of the normal distributed extra white space insertion for the entire line. The intention is that stretching soak up all the excess white space inserted by *dtroff* to adjust the line. If there are no opportunities for stretching or there are too few to soak up all the excess white space, what is not soaked up is distributed in between the words according to *dtroff*'s algorithm. There are several kinds of stretching, and which is in effect for all −**a** designated fonts is specified with the −**s** argument, described below. If it is desired not to stretch a particular Arabic, Hebrew, Persian, or other font, while still stretching others, then the particular font should not be listed in the **−a***`font-position-list`*. Words in such fonts will not be stretched and will be spread with extra white space if the containing line is spread with extra white space. The **−r** and the **−a** specifications are independent. If a font is in the **−a***`font-position-list`* but not in the **−r***`font-position-list`*, then its text will be stretched but not reversed. This independence can be used to advantage when it is necessary to designate a particular Arabic, Hebrew, Persian, or other font as left-to-right for examples or to get around the above mentioned limitations in the use of *eqn*, *ideal*, *pic*, or *tbl*.

The kind of stetching to be done for all fonts designated in the **−a***`font-position-list`* is indicated by the **−s** argument. There are two relatively independent dimensions that must be set to describe the stretching, what is stretched and the places that are stretched. A stretch argument is of the form

−s*mp*
or
−sn

where *m* specifies the stretching mode, i.e, what is stretched, and *p* specifies the places that are stretched. The *m* and *p* must be given in that order and with no intervening spaces. The **−sn** means that there is *no* stretching and normal spreading of words is used even in **−a** designated fonts. The choices for the mode *m* are:

l
(letter ell)
In the words designated by the *p*, stretch the last stretchable letter.

c

In the words designated by the *p*, stretch the last connection to a letter.

e

In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later.

b

In the words designated by the *p*, stretch either the last stretchable letter or the last connection to a letter, whichever comes later, and if it is a letter that is stretched and it is a connect-previous letter then also stretch the connection to the letter.

To our knowledge, all Arabic and Persian fonts, have a baseline filler that can be used to achieve the stretching of connections. It is fairly easy for such a filler to be added to any font definition that does not have it, and moreover to make it the character that is addressed by **\(hy**, which is normally the code for the hyphen character. (Therefore no account is taken of the possibility that stretching connections is not possible.) Since Arabic and Persian do not have a hyphen and hyphenation is turned off when in an Arabic or Persian font, it is safe to use **\(hy** to name the filler. Of course, this requires that the width table for Arabic and Persian fonts have an entry for **hy** designating the filler character in the font, for example:

```
hy    15 0 0267   filler
```

Giving the filler character an explicit *dtroff* two-character name allows *dtroff* to deal with it uniformly despite that it might be in a different position in each font.

On the other hand, stretching of letters requires a dynamic font which, by its very nature of not having a constant bitmap for a given font name, point size, and character name, cannot be type 1 (in PostScript terminology) and cannot be a bitmapped font. Therefore, not all Arabic, Hebrew, and Persian fonts support stretching of letters. Moreover, within a dynamic font, not all characters are stretchable. Historically, only characters with strong horizontal components are stretchable, such as those in the stand-alone and connect-previous forms of the *baa* family. Obviously, one cannot stretch totally vertical characters such as *alif*. Therefore, it is necessary to specify by additional information in the *ditroff* width table for a font which characters are stretchable. In the width table for an Arabic, Hebrew, or Persian font, for each character that is not also an ASCII character, i.e., not also a digit or punctuation, and thus is neither connected or stretchable, one specifies after the name, width, ascender-descender information, and code, two additional fields, the connectivity and the stretchability of the character, in that order. The connectivity is either

N	for stand-alone,
A	for connect-after,
P	for connect-previous,
B	for connect-both, or
U	for unconnected (because it is punctuation or a diacritical, etc.),

and the stretchability is either

N	for not stretchable,
S	for stretchable,

Some examples of width table lines are:

```
%      125 2 045    percent

---    55 0 0101    U      N      comma
---    70 0 0105    U      N      hamza

---    129 0 0106   N      S      baa_SA
---    36 2 0102    N      N      alef_SA

---    113 0 0177   A      N      sad_CA
---    66 2 0215    A      S      caf_CA

---    43 2 0225    P      N      alef_CP
---    120 0 0274   P      S      baa_CP

---    53 0 0230    B      N      baa_CB
---    73 2 0261    B      S      caf_CB
```

Recall that `---` in the name field of a character means that it can be addressed only by **`\N'`***n***`'`**, where *n* is the decimal equivalent of the character's code. Only such lines will have the connectivity and stretchability fields.

For a Hebrew font, for which there is no notion of connectivity of letters, and therefore, the position of the letters is irrelevant for deciding stretching, there is only the possibility of stretching letters. Some examples of width table lines for such fonts are:

```
%     132 3 045   percent

---   95 3 0140   U     N     quoteleft=alef
---   92 3 0141   U     S     a=bet
```

Below, “stretchable unit” refers to what is a candidate for stretching according to the mode. The choices for *p*, which specifies places of stretching, are:

`f`

In any line, stretch the last stretchable unit.

`2`

Assuming that the mode is **`b`** (both), in any line, stretch the last two stretchable units, if they are the connection leading to a stretchable connect-previous letter and that letter, and stretch only the last stretchable unit otherwise. If the mode is not **`b`**, then this choice of places is illegal.

`m`*n* or **`m`**

In any line, stretch the last stretchable unit by an amount not exceeding *n* emms. If that does not exhaust the available white space, then stretch the next last stretchable unit by an amount not exceeding *n* emms, and so on until all the available white space is exhausted. If *n* is not given, it is assumed to be **`2.0`**. In general *n* can be any number in floating point format.

a, **ad**, or **al**

In any line, stretch all stretchable units. In this case, the total amount available for stretching is divided evenly over all stretchable units on the line identified according to the mode. Since the units of stretching are the units of device resolution, the amount available might not divide evenly over the number of places. Therefore, it is useful to be able to specify what to do with the remainder of this division. This specification is given as an extension of the stretching argument. The choices are **d** or **l**, with the former indicating that the excess be distributed as evenly as possible to the spaces between words and the latter indicating that the excess be distributed as evenly as possible in stretchable letters that were stretchable units according to the current mode and place. The latter is the default if no choice is specified. The stretched item for the **l** choice must be a letter rather than a connection because only a stretchable letter is stretchable to any small amount that will be the remainder.

In general, the stretch is divided as evenly as possible between all stretchable units in a line. Specificly, in stretch mode **b**, if we have a connection leading to a stretchable connect-previous letter and that letter, then any stretch remainder we have from stretching the connection will be added to the stretch of the letter.

Sometimes, it is desirable to be able to manually stretch connections or letters to achieve special effects, e.g., more balanced stretching or stretching in lines that are not otherwise adjusted, e.g., centered lines. Stretching a connection can be achieved by using the baseline filler character explicitly as many times as necessary to achieve the desired length. Note that the *troff* line drawing function can be used to get a series of adjacent fillers to any desired length, e.g.,

```
\l'2m\(hy'
```

will draw a string of adjacent base-line fillers of length 2 emms.

To achieve stretching of letters, one should immediately preceed, with no intervening white space, the letter to be stretched by

\X'stretch'\h'*n***'**

where *n* is a valid length expression in *troff*'s input language. *ffortid* is prepared to deal with the output from *dtroff* generated by this input to generate output that will cause the letter immediately following it to be stretched by the length specified in *n*. For example,

\X'stretch'\h'1m'\N'70'

will cause the character whose decimal code is 70 to be stretched by 1 emm. The output will fail to have the desired effect if the letter following is not a stretchable letter.

For finer control over stretching, it may be desirable to inhibit automatic stretching on manually stretched connections and letters. In particular, when manual stretching is done on a letter or its connection for balancing purposes, one does not want additional automatic stretching to be done on the same to mess up the balance. Accordingly, two command line flags are provided for this purpose:

–msc
Do not automatically stretch manually stretched connections.

–msl
Do not automatically stretch manually stretched letters.

These flags are understood as eliminating potential stretching places, letters or connections, that were identified on the basis of the stretch mode, **`l`**, **`c`**, **`e`**, or **`b`**. (In the following description, parenthesized text is a comment stating what is true at this point and not what needs to be done.)

For any letter *l* that is a candidate for stretching by the mode,

```
    if both the letter itself and its connection to the previous letter are candidates then

        if either kind of manual stretch is in the letter and that kind of manual stretch cannot be
        stretched additionally, then neither part of l is any longer a candidate;

    otherwise (only the letter itself is a candidate OR only its connection to the previous letter is a
    candidate)

    if the letter itself is a candidate for stretching by the mode,

        if there is manual stretching in the letter and manually stretched letters cannot be
        stretched more, then l is no longer a candidate;

    otherwise (the connection of l is a candidate for stretching by the mode),

        if there is manual stretching in the connection of l to the previous letter and manually
        stretched connections cannot be stretched more, then l is no longer a candidate.
```

FILES

/usr/lib/tmac/tmac.∗	standard macro files
/usr/lib/font/dev∗/∗	device description and font width tables

SEE ALSO

Cary Buchman, Daniel M. Berry, *User’s Manual for Ditroff/Ffortid, An Adaptation of the UNIX Ditroff for Formatting Bi-Directional Text,*
Johny Srouji, Daniel M. Berry, *An Adaptation of the UNIX Ditroff for Arabic Formatting*
troffort(l), ptrn(l)

Appendix C - ffortid Source Files

```
# define s_token            1
# define f_token            2
# define c_token            3
# define C_token            4
# define H_token            5
# define V_token            6
# define h_token            7
# define v_token            8
# define hc_token           9
# define n_token           10
# define w_token           11
# define p_token           12
# define trail_token       13
# define stop_token        14
# define dev_token         15
# define res_token         16
# define init_token        17
# define font_token        18
# define pause_token       19
# define height_token      20
# define slant_token       21
# define newline_token     22
# define PR_token          23
# define PL_token          24
# define D_token           25
# define N_token           26
# define include_token     27
# define control_token     28
# define postscript_begin_token  29
# define psfig_begin_token  30
# define stretch_token     31
```

```
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
%%
s[0-9]+                                     {return (s_token);}
f[0-9]+                                     {return (f_token);}
c.                                          {return (c_token);}
C..                                         {return (C_token);}
N[0-9]+                                     {return (N_token);}
H[0-9]+                                     {return (H_token);}
V[0-9]+                                     {return (V_token);}
h[0-9]+                                     {return (h_token);}
v[0-9]+                                     {return (v_token);}
n[0-9]+" "[0-9]+                            {return (n_token);}
[0-9][0-9].                                 {return (hc_token);}
w                                           {return (w_token);}
p[0-9]+                                     {return (p_token);}
x" "trailer                                 {return (trail_token);}
x" "stop                                    {return (stop_token);}
x" "T" ".+                                  {return (dev_token);}
x" "r(es)?" "[0-9]+" "[0-9]+" "[0-9]+       {return (res_token);}
x" "i(nit)?                                 {return (init_token);}
x" "f(ont)?" "[0-9]+" ".+                   {return (font_token);}
x" "p(ause)?                                {return (pause_token);}
x" "H" "[0-9]+                              {return (height_token);}
x" "S" "[0-9]+                              {return (slant_token);}
"\n"                                        {return (newline_token);}
x" "PR(\\)?                                 {return (PR_token);}
x" "PL(\\)?                                 {return (PL_token);}
x" "X" "PR                                  {return (PR_token);}
x" "X" "PL                                  {return (PL_token);}
x" "X" "p.+(\\)?$                           {return (psfig_begin_token);}
```

```
x" "TS(\\)?                                        {return (control_token);}
x" "TE(\\)?                                        {return (control_token);}
D.*$                                               {return (D_token);}
&.+\\$                                             {return (include_token);}
\%PB(\\)?                                          {return (postscript_begin_token);}
x" "X" "stretch                                    {return (stretch_token);}
%%
```

```
/*******************************************************************
********************************************************************
**                                                                **
**   this file contains the type definition of the internal token **
**   representation structure                                     **
**                                                                **
********************************************************************
*******************************************************************/

 typedef int bool;

 typedef struct tokn
 {
   int          token_type;             /* token type              */
   int          point_size;             /* point size              */
   int          font;                   /* font number             */
   char         font_name[3];           /* font name               */
   bool         lr;                     /* font direction          */
   int          width;                  /* character width         */
   int          vertical_pos;           /* vertical position       */
   int          horizontal_pos;         /* horizontal position     */
   bool         begining;               /* begining of word indicator */
   bool         ending;                 /* end of word indicator   */
   int          fillers_num;            /* how many fillers to put */
   int          filler_width;           /* the filler width        */
   int          manual_stretch_width;   /* manual amount to stretch */
                                        /* letter. Change #1 - harry */
   int          stretch_width;          /* automatic amount to */
                                        /* stretch letter. Change */
                                        /* #1 - harry */
   char         char1;                  /* 1st character of (abs)char token*/
```

```
    char         char2;                      /* 2nd character of (abs)char token*/
    char         char3;                      /* 3rd character of abs char token*/
    struct tokn  *next;                      /* next token                 */
  } TOKENTYPE, *TOKENPTR;


 /**************************************************************************/
 /**************************************************************************/
```

```c

/*
     this file contains general constant and macro definitions
*/

/* all macros dealing with stretching and connections are change #1 - harry */

#define  BEGINING            1
#define  NOT_BEGIN           0
#define  LEFT_TO_RIGHT       1
#define  RIGHT_TO_LEFT       0
#define  END                 1
#define  NOT_END             0
#define  TRUE                1
#define  FALSE               0
#define  NOFILLERS           0
#define  NOSTRETCH           0
#define  ARABIC              1

#define  STRETCHABLE         'S'
#define  NOTSTRETCHABLE      'N'

#define  STANDALONE          'N'
#define  CONNECTAFTER        'A'
#define  CONNECTPREVIOUS     'P'
#define  CONNECTBOTH         'B'
#define  UNCONNECTED         'U'

#define  DUMP_LEX(TOKN)      printf("%s\n",TOKN)
#define  SET_DIRECTION(FNUM,DIR)  direction_table[FNUM] = DIR
#define  FONT_DIRECTION(FNUM)     direction_table[FNUM]
```

```
#define  SET_AR_FONT(FNUM)        arabic_fonts[FNUM] = TRUE
#define  RESET_AR_FONT(FNUM)      arabic_fonts[FNUM] = FALSE
```

```
/*******************************************************************
*******************************************************************
**                                                               **
**   this file contains the type definition of the internal font   **
**   table structure                                             **
**                                                               **
*******************************************************************
*******************************************************************/

 typedef struct fntable
 {
   char         name[3];              /* font name      */
   bool         direction;            /* font direction */
 } TABLENTRY;;


/**************************************************************************/
/**************************************************************************/
```

```
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
/*************************************************************************
**************************************************************************
**                                                                      **
** this file includes routines that dump the current internal line      **
**                                                                      **
**************************************************************************
*************************************************************************/

#include <stdio.h>
#include "TOKEN.h"
#include "lex.h"
#include "macros.h"

#define  max(a,b)       ((a<b) ? b : a)
#define  MAXZWC         100        /* maximum number of respective zero
                                      width characters        */
TOKENPTR new_token();              /* function return type    */


extern   bool arabic_fonts[256];

/* new stretch flags for change #1 - harry */

extern   char stretch_mode;
extern   char stretch_place;
extern   float stretch_amount;
extern   int msc_flag;
extern   int msl_flag;

```

```
/**************************************************************************
***************************************************************************
**                                                                       **
** this routine dumps the current internal line while reversing          **
** the tokens of the specified lr direction                              **
** Was enhanced in order to deal with ZERO width characters, or          **
** ZERO horizontal movements (via the usage of '\z' for example).        **
** It was also optimized and made more efficient.                        **
**                                                                       **
** Johny                                                                 **
***************************************************************************
**************************************************************************/

dump_line(start,end,reverse_lr)

  TOKENPTR *start;
  TOKENPTR *end;
  bool     reverse_lr;

{

  bool     start_of_line = TRUE;

  TOKENPTR tmptr;
  TOKENPTR new_start = NULL;      /* ptr to start of new token line */
  TOKENPTR new_end   = NULL;      /* ptr to end of new token line   */
  TOKENPTR rev_start = NULL;      /* ptr to start of reversed token line */
  TOKENPTR rev_end   = NULL;      /* ptr to end of reversed token line   */
  TOKENPTR tokenptr  = *start;    /* next token to dump                  */
  TOKENPTR zerowc[MAXZWC];
  int      total_stretch = 0;
```

```
    int      i,j,hpos;

    while(tokenptr != NULL)
    {
      if  (tokenptr->lr != reverse_lr)
      {                             /* if the language is written in it’s
                                       natural direction                 */
        if (tokenptr->next == NULL)
          tokenptr->ending = NOT_END;
        simple_add_token(tokenptr, &new_start, &new_end);/* change #1 - */
                           /* harry */
        tokenptr = tokenptr->next;
      }
      else
      {
        while ((tokenptr != NULL) && (tokenptr->lr == reverse_lr))
        {
          if (tokenptr->ending) {            /* remark word endings */
              tokenptr->ending = NOT_END;
              tokenptr->begining = BEGINING;
          }
          if (tokenptr->begining) {
              tokenptr->ending = END;
              tokenptr->begining = NOT_BEGIN;
          }
          hpos = tokenptr->horizontal_pos;
          i = 0;
          while ((tokenptr->next != NULL) &&
                 (tokenptr->next->horizontal_pos == hpos))
          {
              zerowc[i++] = tokenptr;
```

```
                tokenptr = tokenptr->next;
            }
            tmptr = tokenptr->next;
            if (i>0)        /* a ZERO-width character was detected */
            {
              int temp;

              zerowc[i] = tokenptr;
              for (j=i; j>=0; j--) {
                zerowc[j]->width =
                  tokenBasicWidth(tokenptr); /*change #1 - harry */
                push_token(zerowc[j],&rev_start,&rev_end);
              }
              temp = zerowc[i]->ending;
              zerowc[i]->ending = zerowc[0]->ending;
              zerowc[0]->ending = NOT_END;
              zerowc[0]->begining = temp;
            }
            else push_token(tokenptr,&rev_start,&rev_end);
            tokenptr = tmptr;
          }

          recalculate_horizontal(rev_start,rev_end);

          if (tokenptr == NULL)            /* unmark word end at end of line */
            rev_end->ending = NOT_END;

          tmptr = rev_start;      /* dump reversed line */
          while (tmptr != NULL)
          {
             simple_add_token(tmptr, &new_start, &new_end);/* change #1 - */
```

```
                                                         /* harry */
          tmptr = tmptr->next;
       }

      free_line(&rev_start,&rev_end);
     }
   }

   /* Change #1 - Harry */

   /* stretch Arabic words and reposition letters .. Johny */
   total_stretch = calculate_total_stretching(new_start,new_end);
   if ((total_stretch < 0) ||      /* the line was shrinked!! */
       (number_of_stretch_candidates(new_start,new_end) == 0))
                                   /* nothing to stretch! */
     total_stretch = 0;
   stretch_line(new_start,new_end,total_stretch);

   tokenptr = new_start;
   while(tokenptr != NULL)
   {
      put_token(tokenptr,start_of_line);
      start_of_line = FALSE;
      tokenptr = tokenptr->next;
   }

   free_line(start,end);
   free_line(&new_start,&new_end);

   return;
 }
```

```

/*************************************************************************
**************************************************************************
**                                                                      **
** this routine reverses the specified internal line                    **
** It also recognizes ZERO width characters and preserves their         **
** order with the next letter.                                          **
**                                                                      **
** Johny                                                                **
**************************************************************************
*************************************************************************/


reverse_line(start,end,paper_width)

TOKENPTR *start;
TOKENPTR *end;
int      paper_width;

{
  int       line_length = (*end)->horizontal_pos +
                   tokenBasicWidth(*end); /*change #1 - harry */
  int       new_indent  = paper_width - line_length - 1;
  int       adjustment  = new_indent - (*start)->horizontal_pos;
  TOKENPTR  tmptr = (*start);
  TOKENPTR  save_start;
  TOKENPTR  tmp_start   = NULL;
  TOKENPTR  tmp_end     = NULL;
  bool      begining;
  bool      ending;
  TOKENPTR  zerowc[MAXZWC];
```

```
    int         i,j,hpos;

    (*end)->ending = END;

    while ((tmptr = (*start)) != NULL)
    {
      *start = (*start)->next;

      begining = tmptr->begining;
      ending   = tmptr->ending;
      tmptr->begining = ending;
      tmptr->ending   = begining;
      hpos = tmptr->horizontal_pos;
      i = 0;
     /* enter the equaly hpos tokens into an array */
      while ((tmptr->next != NULL) && (tmptr->next->horizontal_pos == hpos))
      {
          zerowc[i++] = tmptr;
          tmptr = tmptr->next;
      }
      if (i>0) {
          int temp;

          zerowc[i] = tmptr;
          for (j=i-1; j>=0; j--)      /* update the start pointer */
              *start = (*start)->next;
          save_start = *start;
        /* update the width for each one of them */
          for (j=i; j>=0; j--) {
              zerowc[j]->width = tokenBasicWidth(tmptr);
                                  /* take the last token */
```

```
                                                    /* width */
                                                /*change #1 - harry */

                  zerowc[j]->horizontal_pos += adjustment;
                  push_token(zerowc[j],&tmp_start,&tmp_end);
              }
              temp = zerowc[i]->ending;
              zerowc[i]->ending = zerowc[0]->ending;
              zerowc[0]->ending = NOT_END;
              zerowc[0]->begining = temp;
              *start = save_start;
          }
          else if (tmptr != NULL) {
              tmptr->horizontal_pos += adjustment;
              push_token(tmptr,&tmp_start,&tmp_end);
          }
        }

        *start = tmp_start;
        *end   = tmp_end;

        recalculate_horizontal(*start,*end);

        return;
      }

      /**************************************************************************
      ***************************************************************************
      **                                                                       **
      ** this routine recalculates the horizontal motion in the specified      **
      ** internal line                                                         **
```

```
**                                                                      **
************************************************************************
***********************************************************************/

recalculate_horizontal(start,end)

  TOKENPTR  start;                  /* ptr to start of line */
  TOKENPTR  end  ;                  /* ptr to end of line   */

{

  TOKENPTR prev_token;
  TOKENPTR next_token;
  int      new_horizontal;
  int      prev_horizontal;
  int      start_horizontal;

  start_horizontal = start->horizontal_pos;
  prev_token = start;
  prev_horizontal = end->horizontal_pos;
  new_horizontal  = prev_horizontal;

  if(prev_token->next != NULL)
  {
     next_token = prev_token->next;
     while (next_token != NULL)
     {
        /*change #1 - harry */

       new_horizontal = new_horizontal + (prev_token->horizontal_pos -
                                next_token->horizontal_pos) +
                        tokenBasicWidth(prev_token)
                 - tokenBasicWidth(next_token);
         prev_token->horizontal_pos = prev_horizontal;
         prev_token = next_token;
         prev_horizontal = new_horizontal;
         next_token = prev_token->next;
       }
     }
     prev_token->horizontal_pos = new_horizontal;
   }

/**************************************************************************
***************************************************************************
**                                                                       **
** This function should return the total stretching amount.              **
** It does that, by calculating the total amount of extra interword      **
** spaces.                                                               **
**                                                                       **
** Johny                                                                 **
***************************************************************************
**************************************************************************/

int
calculate_total_stretching (start,end)

TOKENPTR  start;                  /* ptr to start of line */
TOKENPTR  end  ;                  /* ptr to end of line   */

{
  TOKENPTR prev_token;
  TOKENPTR next_token;
  TOKENPTR tokenptr  = start;
  int      total_stretch = 0;
  int      space_width;

  if ((tokenptr != NULL) && (tokenptr->next != NULL))
  {
      prev_token = tokenptr;
      next_token = tokenptr->next;
      while (next_token != NULL)
      {
        /* we have to calculate the space width for each word, as the
           point size can change at any point */
         /* rounding error fix. change #1 - harry */

        space_width = width1(0, prev_token->point_size,
                   prev_token->font);

        if (prev_token->ending) {
           /*change #1 - harry */

           total_stretch = total_stretch +
             (next_token->horizontal_pos -
              prev_token->horizontal_pos) -
                (tokenBasicWidth(prev_token)
            + space_width);
        }
        prev_token = next_token;
        next_token = prev_token->next;
      }
      tokenptr = next_token;
  }
```

```

   return(total_stretch);
}


/* All the following functions were changed for change #1 - harry */
/* Some of them are based on previous functions and some are completly */
/* new */



/**************************************************************************
***************************************************************************
**                                                                       **
** stretch a line by 'total_stretch' according to the current stretch   **
** mode and place.                                                       **
**                                                                       **
***************************************************************************
**************************************************************************/

stretch_line (start,end,total_stretch)
TOKENPTR start;           /* ptr to start of line */
TOKENPTR end;             /* ptr to end of line   */
int total_stretch;
{
   int remainder = 0;
   int candidates;

   if ((total_stretch == 0) || (stretch_mode == 'n'))
      return;

```

```
    switch (stretch_place) {
    case 'f':
    case '2':
    case 'm':
       {
          bool finished = FALSE;
          TOKENPTR tokenptr = start;
          bool letter_stretch;
          bool connect_stretch;
          TOKENPTR token;

          while ((tokenptr != NULL) && !finished) {
             if (candidates = word_stretch_candidates(tokenptr,
                              &letter_stretch,
                              &connect_stretch, & token)) {
                if (stretch_place == '2')
                  remainder = stretch_word(tokenptr,
                               total_stretch/candidates, TRUE);
                else remainder = stretch_word(tokenptr,
                                  total_stretch, TRUE);
                if (stretch_place == 'm')
                  total_stretch = remainder;
                if ((stretch_place != 'm') || (remainder == 0)) {
                   recalculate_spaces(start,end,remainder);
                   finished = TRUE;
                }
             }
             while ((tokenptr != NULL) && !(tokenptr->ending))
               tokenptr = tokenptr->next;
             if (tokenptr != NULL)
               tokenptr = tokenptr->next;
```

```
                if (!finished && (tokenptr == NULL) &&
                    (stretch_place == 'm')) {
                    recalculate_spaces(start,end,remainder);
                    finished = TRUE;
                }
            }
            break;
        }

    case 'a':
    case 'd':
        remainder = spread_stretch_in_line(start, end, total_stretch);
        recalculate_spaces(start,end,remainder);
        break;
    };
    return;
}


/**************************************************************************
 **************************************************************************
 **                                                                      **
 ** recalculates the inter words spaces, after stretching a specif-      **
 ** ied letter. It simply removes the  extra spaces added by  troff      **
 ** and spread the amount of remainder, which is the amount of poi-      **
 ** nts less that the filler width.                                      **
 ** initialization: hmove = 0                                            **
 ** when reaching an end of word: hmove = hmove - extra_space            **
 ** when reaching a letter with a filler: hmove = hmove + filler         **
 **                                                                      **
 ** Johny.                                          **
 ** Modified to handle letter stretching as well.                        **
 **                                                                      **
 **************************************************************************
 *************************************************************************/

 recalculate_spaces (start,end,remainder)

 TOKENPTR  start;                   /* ptr to start of line */
 TOKENPTR  end  ;                   /* ptr to end of line   */
 int       remainder;

 {
   TOKENPTR prev_token;
   TOKENPTR next_token;
   TOKENPTR tokenptr  = start;
   int      space_width=0, filler_width=0;
   int      extra_space=0, hmove=0;
   int      prev_horizontal;
   int      nw, rpw=0;                  /* remainder per word   */
   bool     last_word = TRUE;

   if ((nw = number_of_words(start,end)) == 0)
     return;
   if (nw == 1)
    rpw = 0;
   else rpw = (int) (remainder / (nw - 1));

   if ((tokenptr != NULL) && (tokenptr->next != NULL))
   {
       prev_token = tokenptr;
       next_token = tokenptr->next;
```

```
          prev_horizontal = prev_token->horizontal_pos;
          while (next_token != NULL)
          {
            space_width = width1(0, prev_token->point_size,
                       prev_token->font);
            filler_width = width2("hy",
                   next_token->point_size,
                   next_token->font);
            if (prev_token->ending) {
               extra_space = (next_token->horizontal_pos -
                    prev_horizontal) -
                      (tokenFullWidth(prev_token)
                       + space_width);
               hmove = hmove - extra_space + rpw;
               if (last_word) {
                  if (nw > 1)
                    hmove = hmove + (remainder % nw);
                  last_word = FALSE;
                }
       } else
         {
            if (prev_token->fillers_num > 0)
              hmove = hmove + prev_token->fillers_num * filler_width;
            if (prev_token->stretch_width > 0)
              hmove += prev_token->stretch_width;
         }
            prev_horizontal = next_token->horizontal_pos;
            next_token->horizontal_pos += hmove;
            prev_token = next_token;
            next_token = next_token->next;
         }
```

```
        tokenptr = next_token;
    }
}


/**********************************************************************
***********************************************************************
**                                                                   **
** return the number of words in a line.                             **
** Johny.                                                            **
**                                                                   **
***********************************************************************
**********************************************************************/

int number_of_words (start,end)
TOKENPTR  start;                  /* ptr to start of line */
TOKENPTR  end  ;                  /* ptr to end of line   */
{
  TOKENPTR  tokenptr = start;
  int       nw = 0;               /* the number of words in line */

  while (tokenptr != NULL) {
    if (tokenptr->ending)
        nw++;
    else if (tokenptr->next == NULL)
      nw++;
    tokenptr = tokenptr->next;
  }
  return(nw);
}
```

```

/*************************************************************************
**************************************************************************
**                                                                      **
** return the number of stretch candidates in a word according to the   **
** current stretch mode and a pointer to a token that is to stretched   **
** including flags indicating if the letter or connection are to be     **
** stretched.                                                           **
**                                                                      **
**************************************************************************
*************************************************************************/

int word_stretch_candidates (start, letter_stretch_flag,
              connect_stretch_flag, token)
TOKENPTR  start;                /* ptr to start of word */
bool* letter_stretch_flag;      /* token should be letter stretched */
bool* connect_stretch_flag;     /* token should be connection stretched */
TOKENPTR* token;                /* token to be stretched if found */
{
   TOKENPTR tokenptr = start;
   bool finished = FALSE;
   char letter[4];
   int i;
   int candidates = 0;

   *letter_stretch_flag = FALSE;
   *connect_stretch_flag = FALSE;
   while ((tokenptr != NULL) && !finished) {
      if (arabic_fonts[tokenptr->font] == ARABIC) {
         sprintf(letter,"%c%c%c",tokenptr->char1,tokenptr->char2,
            tokenptr->char3);
```

```
              i = atoi(letter);
              if ((stretch_mode == 'l') || (stretch_mode == 'e')) {
                 if (stretchable(i, tokenptr->font) &&
                     !(msl_flag &&
                       (tokenptr->manual_stretch_width
                        > 0))) {
                    *letter_stretch_flag = TRUE;
                    *token = tokenptr;
                    finished = TRUE;
                    candidates = 1;
                 }
              }
              if (((stretch_mode == 'c') || (stretch_mode == 'e'))
                  && !finished) {
                 if (connectable(i, tokenptr->font) &&
                     !(msc_flag &&
                       (tokenptr->next != NULL) &&
                       (tokenptr->next->token_type == C_token) &&
                       (tokenptr->next->char1 == 'h') &&
                       (tokenptr->next->char2 == 'y'))) {
                    *connect_stretch_flag = TRUE;
                    *token = tokenptr;
                    finished = TRUE;
                    candidates = 1;
                 }
              }
              if (stretch_mode == 'b') {
                 if (stretchable(i, tokenptr->font) &&
                     !(msl_flag &&
                        (tokenptr->manual_stretch_width
                    > 0)) &&
                     !(msc_flag &&
                       (tokenptr->next != NULL) &&
                       (tokenptr->next->token_type == C_token) &&
                       (tokenptr->next->char1 == 'h') &&
                       (tokenptr->next->char2 ==
                        'y'))) {
                    *letter_stretch_flag = TRUE;
                    *token = tokenptr;
                    finished = TRUE;
                    candidates = 1;
                 }
                 if (connectable(i, tokenptr->font) &&
                     !(msl_flag &&
                        (tokenptr->manual_stretch_width
                    > 0)) &&
                     !(msc_flag &&
                       (tokenptr->next != NULL) &&
                       (tokenptr->next->token_type == C_token) &&
                       (tokenptr->next->char1 == 'h') &&
                       (tokenptr->next->char2 ==
                        'y'))) {
                    *connect_stretch_flag = TRUE;
                    *token = tokenptr;
                    finished = TRUE;
                    candidates += 1;
                 }
              }
           }
           if (tokenptr->ending)
             finished = TRUE;
           else tokenptr = tokenptr->next;
```

```
   }

   return candidates;
}


/**********************************************************************
***********************************************************************
**                                                                   **
** return the number of stretch candidates in a line according to the   **
** current stretch mode.                                             **
**                                                                   **
***********************************************************************
**********************************************************************/

int number_of_stretch_candidates (start,end)
TOKENPTR  start;                   /* ptr to start of line */
TOKENPTR  end  ;                   /* ptr to end of line   */
{
   TOKENPTR  tokenptr = start;
   bool letter_stretch;
   bool connect_stretch;
   TOKENPTR token;
   int       nsc = 0;        /* the number of stretch candidates in line */

   while (tokenptr != NULL) {
      nsc += word_stretch_candidates(tokenptr,
                          &letter_stretch,
                          &connect_stretch, &token);
      while ((tokenptr != NULL) && !(tokenptr->ending))
        tokenptr = tokenptr->next;
        if (tokenptr != NULL)
          tokenptr = tokenptr->next;
    }

    return nsc;
}


/*************************************************************************
**************************************************************************
**                                                                      **
** stretches a letter by 'stretch' and returns the remainder if any.    **
** if the 'do_stretch' flag is false no actual stretching is preformed. **
**                                                                      **
**************************************************************************
*************************************************************************/

int letter_stretch (tokenptr, stretch, do_stretch)
TOKENPTR tokenptr;
int stretch;
bool do_stretch;
{
   if (do_stretch)
      tokenptr->stretch_width += stretch;
   return 0;
}


/*************************************************************************
**************************************************************************
**                                                                      **
** stretches a connection to a letter by 'stretch' and returns the      **
** remainder if any.                                                    **
** if the 'do_stretch' flag is false no actual stretching is preformed. **
**                                                                      **
**************************************************************************
*************************************************************************/

int connect_stretch (tokenptr, stretch, do_stretch)
TOKENPTR tokenptr;
int stretch;
bool do_stretch;
{
   int filler_width;

   filler_width = width2("hy", tokenptr->point_size, tokenptr->font);
   if (do_stretch) {
      tokenptr->filler_width = filler_width;
      tokenptr->fillers_num += (int) (stretch / filler_width);
   }

   return (stretch % filler_width);
}


/*************************************************************************
**************************************************************************
**                                                                      **
** stretches a word by 'stretch' according to the current stretch mode  **
** and place and returns the remainder if any.                          **
** if the 'do_stretch' flag is false no actual stretching is preformed. **
**                                                                      **
```

```
 ***************************************************************************
 ***************************************************************************/

int stretch_word(start, stretch, do_stretch)
TOKENPTR start;
int stretch;
bool do_stretch;
{
   bool letter_stretch_flag;
   bool connect_stretch_flag;
   TOKENPTR tokenptr;
   int candidates;
   int remainder;
   int temp;
   int st;
   int st_amount;     /* stretch amount in emms in 'm' mode */

   if (stretch == 0)
     return 0;

   candidates = word_stretch_candidates(start, &letter_stretch_flag,
                     &connect_stretch_flag, &tokenptr);

   if (candidates == 0)   /* can't stretch this word */
     return stretch;

   st_amount = (stretch_amount * 6 * width2("\\|", tokenptr->point_size,
                    tokenptr->font));

   if (stretch_place == 'f') {  /* make sure only one candidate */
                 /* is stretched */
```

```
            if (letter_stretch_flag == TRUE)
              connect_stretch_flag = FALSE;
         }

         remainder = 0;
         if (connect_stretch_flag) {
            if (stretch_place == 'm') {
               if (stretch > st_amount)
                 st = st_amount;
               else st = stretch;
               remainder = stretch+connect_stretch(tokenptr,st,do_stretch)-st;
            }
            else remainder = connect_stretch(tokenptr,stretch,do_stretch);
            if (stretch_place == 'm')
              stretch = remainder;
            else stretch += remainder;
         }
         if (letter_stretch_flag) {
            if (stretch_place == 'm') {
               if (stretch > st_amount)
                 st = st_amount;
               else st = stretch;
               remainder = stretch+letter_stretch(tokenptr,st,do_stretch)-st;
            }
            else remainder = letter_stretch(tokenptr, stretch, do_stretch);
         }

         return remainder;
      }


```

```
/***************************************************************************
****************************************************************************
**                                                                        **
** returns the total remainders from all of the stretchable candidates   **
** assuming they are stretched with stretch 'spc'                         **
**                                                                        **
****************************************************************************
***************************************************************************/

int stretch_remainders (start,end,spc)

TOKENPTR start;        /* ptr to start of line */
TOKENPTR end ;           /* ptr to end of line   */
int spc;          /* stretch per candidate */
{
   TOKENPTR  tokenptr = start;
   int remainder = 0;
   bool letter_stretch_flag;
   bool connect_stretch_flag;
   TOKENPTR temp;

   while (tokenptr != NULL) {
      if (word_stretch_candidates(tokenptr,
                     &letter_stretch_flag,
                     &connect_stretch_flag,
                     &temp) > 0)
        remainder += stretch_word(tokenptr, spc, FALSE);
      while ((tokenptr != NULL) && !(tokenptr->ending))
        tokenptr = tokenptr->next;
      if (tokenptr != NULL)
        tokenptr = tokenptr->next;
```

```
    }

    return remainder;
}

/**************************************************************************
***************************************************************************
**                                                                       **
** stretches all the stretchable units in the line according to the     **
** stretch mode and place by dividing the total stretch evenly among    **
** them.                                                                 **
**                                                                       **
***************************************************************************
**************************************************************************/

int spread_stretch_in_line (start,end,total_stretch)
TOKENPTR start;            /* ptr to start of line */
TOKENPTR end  ;            /* ptr to end of line   */
int total_stretch;         /* total stretch to spread in line */
{
   int nc = 0;             /* number of candidates */
   int spc = 0;               /* stretch per candidate */
   int last_word_remainder;
   bool last_word = TRUE;
   TOKENPTR tokenptr = start;
   int candidates;
   int remainder;
   int letter_candidates = 0;    /* letter candidates in 'a' */
                           /* stretch place */
   int splc = 0;        /* stretch per letter candidate in 'a' */
                     /* place */
   int splc_extra = 0;     /* stretch per letter candidate in 'a' */
                     /* place extra only in some words */
   char temp;
   bool letter_stretch;
   bool connect_stretch;
   TOKENPTR token;

   if ((nc = number_of_stretch_candidates(start, end)) == 0)
      return total_stretch;

   spc = (int) (total_stretch / nc);

   remainder = (total_stretch % nc) +
     stretch_remainders(start,end, spc);

   if ((stretch_place == 'a') && (remainder > 0) &&
          (stretch_mode != 'c')) {
         temp = stretch_mode;
         stretch_mode = 'l';
         letter_candidates =
           number_of_stretch_candidates(start, end);
         stretch_mode = temp;
         if (letter_candidates > 0) {
           splc = (int) (remainder / letter_candidates);
           if (remainder % letter_candidates > 0)
             splc_extra = 1;
        }
      }

   while (tokenptr != NULL) {
      candidates = word_stretch_candidates(tokenptr,
                            &letter_stretch,
                            &connect_stretch,
                            &token);
          if (candidates > 0) {
             stretch_word(tokenptr, spc, TRUE);
             if (letter_candidates > 0) {
                temp = stretch_mode;
                stretch_mode = 'l';
                candidates = word_stretch_candidates(tokenptr,
                                  &letter_stretch,
                                  &connect_stretch,
                                  &token);
                if ((candidates > 0) && (remainder > 0)) {
                   stretch_word(tokenptr,
                           splc + splc_extra, TRUE);
                   remainder -= splc*candidates + splc_extra;
                }
                stretch_mode = temp;
             }
          }
          while ((tokenptr != NULL) && !(tokenptr->ending))
            tokenptr = tokenptr->next;
          if (tokenptr != NULL)
            tokenptr = tokenptr->next;
       }

       return remainder;
    }


    /* used for debugging ... Johny */
```

```

print_line(start,end)
TOKENPTR  start;                 /* ptr to start of line */
TOKENPTR  end  ;                 /* ptr to end of line   */
{

  TOKENPTR tmptr = start;

  while (tmptr != NULL) {
    printf("horiz:%d , vert:%d , manual stretch:%d, stretch:%d fillers:%d char:%c%c%c , width:%d\n",
            tmptr->horizontal_pos,tmptr->vertical_pos,
            tmptr->manual_stretch_width, tmptr->stretch_width,
            tmptr->fillers_num,tmptr->char1,
         tmptr->char2,tmptr->char3,tmptr->width);
    tmptr = tmptr->next;
  }
  printf("\n");
}
```

```c
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */
/*********************************************************
**********************************************************
**                                                      **
** this file contains a number of routines that work on **
** internal tokens and token lines                      **
**                                                      **
**********************************************************
*********************************************************/

#include  <stdio.h>
#include  "TOKEN.h"
#include  "TABLE.h"
#include  "macros.h"
#include  "lex.h"



extern int  out_font;                   /* current output font */
extern int  out_size;                   /* current output size */
extern int  out_horizontal;             /* current output horizontal position */
extern int  out_vertical;               /* current output vertical position */
extern char out_font_name[];            /* current output font name */
extern TABLENTRY out_fontable[];        /* current output font table */
extern int  in_font;                    /* current input font */
extern int  in_size;                    /* current input size */
extern int  in_horizontal;              /* current input horizontal position */
extern int  in_vertical;                /* current input vertical position */
extern char in_font_name[];             /* current input font name */
extern TABLENTRY in_fontable[];         /* currnet input font table */
extern bool in_lr;                      /* current input font direction */
extern bool direction_table[];          /* direction of font i */

/****************************************************************
*****************************************************************
**                                                             **
** this routine allocates and initializes a new internal token **
**                                                             **
*****************************************************************
*****************************************************************/

TOKENPTR
new_token (toktyp,ptsize,font,horizontal,vertical,fontdir,beg_wrd,end_wrd,char1,
           char2,char3,f_name,width,manual_stretch)

  int         toktyp;         /* token type                  */
  int         ptsize;         /* point size                  */
  int         font;           /* font number                 */
  int         horizontal;     /* horizontal position         */
  int         vertical;       /* vertical position           */
  bool        fontdir;        /* font direction              */
  bool        beg_wrd;        /* begining of word indicator */
  bool        end_wrd;        /* end of word indicator       */
  char        char1;          /* 1st char of (abs) character token*/
  char        char2;          /* 2nd char of (abs) character token*/
  char        char3;          /* 3nd char of abs character token*/
  char        *f_name;        /* font name                   */
  int         width;          /* character width             */
  int         manual_stretch; /* manual letter stretch. change #1 - harry */

{

  char  *calloc();
  TOKENPTR tokenptr = (TOKENTYPE *) calloc(1,sizeof(TOKENTYPE));

  if (tokenptr != NULL)
  {
    tokenptr->token_type    = toktyp;
    tokenptr->point_size    = ptsize;
    tokenptr->font          = font;
    tokenptr->horizontal_pos= horizontal;
    tokenptr->vertical_pos  = vertical;
    tokenptr->lr            = fontdir;
    tokenptr->begining      = beg_wrd;
    tokenptr->ending        = end_wrd;
    tokenptr->fillers_num  = NOFILLERS;   /* initialization ... Johny */
    tokenptr->manual_stretch_width = manual_stretch; /* change #1 - */
                                         /* harry */
    tokenptr->stretch_width = NOSTRETCH;     /* default is no */
                                 /* stretch. change #1 - */
                                 /* harry */
    tokenptr->char1         = char1;
    tokenptr->char2         = char2;
    tokenptr->char3         = char3;
    strcpy(tokenptr->font_name,f_name);
    tokenptr->width         = width;
    tokenptr->next          = NULL;
  }
  else
  {
    out_of_memory();
  }
    return (tokenptr);
}


/***********************************************************************/
/***********************************************************************/


/*
    this routine returns the basic (before stretch) width of a token
                                                      */

int tokenBasicWidth (tokenptr)
TOKENPTR tokenptr;
{
   return tokenptr->width + tokenptr->manual_stretch_width;
}


/***********************************************************************/
/***********************************************************************/


/*
    this routine returns the full (including stretch) width of a token
                                                      */

int tokenFullWidth (tokenptr)
TOKENPTR tokenptr;
{
   return tokenBasicWidth(tokenptr) + tokenptr->stretch_width;
```

```
}


/*************************************************************************/
/*************************************************************************/


/*
    this routine returns the total stretch (including manual) of a token
                                                    */

int tokenStretch (tokenptr)
TOKENPTR tokenptr;
{
   return tokenptr->stretch_width + tokenptr->manual_stretch_width;
}


/*************************************************************************/
/*************************************************************************/


/*
    this routine adds a token to the end of a line
                                                    */

add_token (tokenptr,start,end)

  TOKENPTR     tokenptr;           /* token to be added        */
  TOKENPTR     *start;             /* ptr to start of line ptr */
  TOKENPTR     *end;               /* ptr to end of line ptr   */
```

```

{
  if (*start == NULL)
    *start = tokenptr;
  else {
    (*end)->next = tokenptr;
   tokenptr->next = NULL;        /* better to make sure it's NULL */
  }
  *end = tokenptr;
  return;
}

/***************************************************************************/
/***************************************************************************/

/*
    this routine adds a token to the end of a line without changing
    the tokenptr
                                                    */

simple_add_token (tokenptr,start,end)

  TOKENPTR     tokenptr;           /* token to be added        */
  TOKENPTR     *start;             /* ptr to start of line ptr */
  TOKENPTR     *end;               /* ptr to end of line ptr   */

{
  if (*start == NULL)
    *start = tokenptr;
  else (*end)->next = tokenptr;
  *end = tokenptr;
  return;
}

/**********************************************************************/
/**********************************************************************/

/*
    this routine pushes a token onto the front of a line
                                                              */
push_token (tokenptr,start,end)

  TOKENPTR     tokenptr;            /* token to be push          */
  TOKENPTR     *start;              /* ptr to start of line ptr */
  TOKENPTR     *end;                /* ptr to end of line ptr   */

{

  if (*end == NULL) {
                               /* reset the prev and next pointers ...
                                  it's not sure that we call it ONLY
                                  after new_token.  Johny  */
   (tokenptr)->next = NULL;
    *end = tokenptr;
  }
  else tokenptr->next = (*start);
  *start = tokenptr;
  return;
}

/******************************************
 ******************************************
```

```
**                                        **
** this routine outputs an internal token **
**                                        **
******************************************
*****************************************/

put_token (tokenptr,start_of_line)

  TOKENPTR tokenptr;           /* token to output */
  bool     start_of_line;      /* indicates first character in line */

{
  if (start_of_line)
  {
    printf("H%d\n",tokenptr->horizontal_pos);
  }

  if (tokenptr->point_size != out_size)
  {
    printf ("s%d\n",tokenptr->point_size);
    out_size = tokenptr->point_size;
  }
  if (strcmp(tokenptr->font_name,out_font_name))
  {
    if (strcmp(tokenptr->font_name,out_fontable[tokenptr->font].name))
    {
      printf("x font %d %s\n",tokenptr->font,tokenptr->font_name);
      new_font(tokenptr->font,tokenptr->font_name,tokenptr->lr,
               out_fontable);
    }
    printf ("f%d\n",tokenptr->font);
```

```
      out_font = tokenptr->font;
      strcpy(out_font_name,tokenptr->font_name);
    }

    if (tokenptr->vertical_pos != out_vertical)
    {
      printf ("V%d\n",tokenptr->vertical_pos);
      out_vertical = tokenptr->vertical_pos;
    }

    if (!start_of_line)
    {
      if (tokenptr->horizontal_pos < out_horizontal)
      {
        printf("H%d\n",tokenptr->horizontal_pos);
      }
      else if (tokenptr->horizontal_pos > out_horizontal) {
        int j =(tokenptr->horizontal_pos - out_horizontal);
        if (j < 100 && j > 9 && tokenptr->token_type == c_token){
                printf( "%d%c", j, tokenptr->char1);
                goto compress;
            }
        printf("h%d",(tokenptr->horizontal_pos - out_horizontal));
      }
    }


  /* the following lines have changes to handle letter stretching. */
  /* Change # 1 - harry */

    switch (tokenptr->token_type) {
      case c_token: printf("c%c\n",tokenptr->char1); break;
      case C_token: printf("C%c%c\n",tokenptr->char1,tokenptr->char2); break;
      case N_token:
        if (tokenStretch(tokenptr) == 0)
     printf("N%c%c%c\n",tokenptr->char1,tokenptr->char2,
            tokenptr->char3);
        else {
     printf("x X stretch %d\n",tokenStretch(tokenptr));
     printf("N%c%c%c\n",tokenptr->char1,tokenptr->char2,
            tokenptr->char3);
        }
        break;
    }
  compress: /* thanks to Mulli Bahr hnncx compressed to nnx */
    out_horizontal = tokenptr->horizontal_pos;

    /* added to insert the fillers ... Johny */
    if (tokenptr->fillers_num > 0) {
     int i;
     printf("h%d",tokenFullWidth(tokenptr));
     printf("Chy\n");
     for (i=0; i<(tokenptr->fillers_num)-1; i++)
        printf("h%dChy\n",tokenptr->filler_width);
     printf("h%d",tokenptr->filler_width);
     out_horizontal += tokenptr->fillers_num * tokenptr->filler_width +
       tokenFullWidth(tokenptr);
    }

    if (tokenptr->ending == END)
      printf("w");

    return;
  }


  /**************************************************************
  ***************************************************************
  **                                                           **
  ** this routine outputs a new page token and its associated **
  ** font and point size tokens and resets its font and size  **
  ** to null and 0 so as to force output of f and s commands  **
  **                                                           **
  ***************************************************************
  **************************************************************/

  put_page_token(page_token)

    char  *page_token;          /* lex page token value */

  {

    extern char  yytext[];     /* lex token value */

    printf("V%d\n",in_vertical);
              /* ditroff guarantees and thus drivers
              require that before page_token, a V0 token!
              Mulli Bahr, Oct, 86*/
    out_vertical = in_vertical;
    printf("%s\n",page_token);

    out_size = 0;                              /* reset out_size to zero ... */
    out_font_name[0] = '\0';                   /* reset out_font_name to empty
```

```
              string  so that the next token print will be forced to dump out s and f
              commands. On some devices the current size and page are not remembered
              across page boundaries, especially if there's a bunch of "x font XX"s.
              Besides the ditroff output always seems to have the s and f issued just
              before the text on each page */

  return;
}


/************************************************************************/
/************************************************************************/


/*
    this routine deallocates a line of tokens
                                                  */

free_line(start,end)

  TOKENPTR  *start;           /* ptr to start of line */
  TOKENPTR  *end;             /* ptr to end of line   */

{
  TOKENPTR  tokenptr;         /* next element in line */

  while( (tokenptr = *start) != NULL)
  {
    *start = tokenptr->next;
    cfree(tokenptr);
  }
```

```
    *end = NULL;
    return;
}

/**********************************************************************/
/**********************************************************************/
```

```
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */

/*******************************************************
 *******************************************************
 **                                                   **
 ** this file includes the main ffortid driver routine **
 **                                                   **
 *******************************************************
 *******************************************************/

#include <stdio.h>
#include "TOKEN.h"
#include "TABLE.h"
#include "lex.h"
#include "lexer"
#include "macros.h"

/*************************************************************************/
/*************************************************************************/

#define  MARK_PREVIOUS_END   in_end->ending = END

#define  ADD_CHAR1(TOKTYP,BEGN,CHAR1) tokenptr = new_token(TOKTYP,in_size,in_font,
in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,NULL,NULL,in_font_name,width1(C
HAR1,in_size,in_font),NOSTRETCH); add_token(tokenptr, &in_start, &in_end )

#define  ADD_CHAR2(TOKTYP,BEGN,CHAR1,CHAR2) tokenptr = new_token(TOKTYP,in_size,in
_font,in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,CHAR2,NULL,in_font_name,w
idth2(yytext+1,in_size,in_font),NOSTRETCH); add_token(tokenptr, &in_start, &in_end
 )
```

```

/*****************************************************************************/
#define  ADD_CHARN(TOKTYP,BEGN,CHAR1,CHAR2,CHAR3,STRETCH) tokenptr = new_token(TOKTYP,in_size,in_font,in_horizontal,in_vertical,in_lr,BEGN,NOT_END,CHAR1,CHAR2,CHAR3,in_font_name,widthn(yytext+1,in_size,in_font),STRETCH); add_token(tokenptr, &in_start, &in_end )

/*****************************************************************************/
/* I think this is better ... Johny                                          */
#define  USAGE  "Usage: %s [-wpaperwidth] [-rfont_position_list] [-afont_position_list] [-s[nfla]]\n"
/*****************************************************************************/

  int   in_font=1;                     /* current input font                 */
  int   in_size=0;                     /* current input size                 */
  int   in_horizontal=0;               /* current input horizontal position  */
  int   in_vertical=0;                 /* current input vertical position    */
  char  in_font_name[3];               /* current input font name            */
  bool  in_lr=LEFT_TO_RIGHT;           /* current input font direction       */
  TABLENTRY in_fontable[256];          /* current input font table           */

  int   out_font=1;                    /* current output font                */
  int   out_size=0;                    /* current output size                */
  int   out_horizontal=0;              /* current output horizontal position */
  int   out_vertical=0;                /* current output vertical position   */
  char  out_font_name[3];              /* current output font name           */
  TABLENTRY out_fontable[256];         /* current output font table          */

  bool  direction_table[256];          /* formatting direction of font i     */
  bool    arabic_fonts[256];           /* whether font i is arabic or not    */

  /* new stretch flags for change #1 - harry */

  char stretch_mode='n';                /* the stretching mode. default */
                                  /* is no stretching */
  char stretch_place='f';               /* the stretching place. no */
                                /* default */
  float stretch_amount=2.0;             /* stretch amount in emms, used */
                                  /* with 'm' stretching place. */
  bool msc_flag=0;                      /* do not automatically stretch */
                                /* manually stretched */
                               /* connections. default is flag */
                                /* off */
  bool msl_flag=0;                      /* do not automatically stretch */
                                /* manually stretched letters */
                                /* default is flag off */

  char   *device;           /* output device             */
  char   c;              /* for flushing included postscript
                        and psfig text    */
  static char copyright[]="(c) Copyright 1987 Berry Computer Scientists, Ltd.";

/*******************************************************************************/
/*******************************************************************************/

main (argc,argv)
  int        argc;
  char          *argv[];

{

  extern char  yytext[];    /* lex token string                */
```

```

              /* indicates the predominate formatting direction */
       bool          lr_predom = LEFT_TO_RIGHT;
       TOKENPTR     in_start  = NULL;       /* ptr to start of internal input line */
       TOKENPTR     in_end    = NULL;       /* ptr to end of internal input line   */

       double       paper_inch=8.5;        /* paper width in inches    */
       int         paper_width;           /* paper width in points     */
       int         token_num;          /* current lex token            */
       bool          new_word=TRUE;        /* indicates word token was encountered*/
       bool          previous_D=FALSE;      /* indicates a D_token was just done   */
       char          small_motion[3];      /* motion from hc_token           */
       char          f_name[3];           /* new font name                */
       int         f_num;              /* new font number           */
       TOKENPTR     tokenptr;                  /* ptr to new internal tokens       */
       TOKENPTR     new_token();              /* ptr to new internal tokens        */
       char          *calloc();
       double       atof();
       int         i,j,k;              /* counter index  */
       char          *tmpyy;                 /* temp ptr for yytext */

     /**************************************************************************/
     /**************************************************************************/

       for (i=0;i<=255;i++) {      /* johny */
         SET_DIRECTION(i,LEFT_TO_RIGHT);
        RESET_AR_FONT(i);
       }

       if (--argc > 0)
       {
```

```
     if (argv[1][0] == '-')
     {
       i=1;
       while(i<= argc)
       {
    switch (argv[i][1])

    {
           /* specify the fonts to be reversed */
      case 'r':
      case 'R':
        {
          if (strlen(argv[i]) > 2)
       SET_DIRECTION(atoi(argv[i++]+2),RIGHT_TO_LEFT);
          else
       i++;
          while( (i<=argc) && (argv[i][0] != '-') )
       SET_DIRECTION(atoi(argv[i++]),RIGHT_TO_LEFT);
          break;
        }
           /* specify the Arabic Fonts. Only Arabic Fonts are stretched */
      case 'a':
      case 'A':
        {
          if (strlen(argv[i]) > 2)
       SET_AR_FONT(atoi(argv[i++]+2));
          else
       i++;
          while( (i<=argc) && (argv[i][0] != '-') )
       SET_AR_FONT(atoi(argv[i++]));
          break;
        }
         /* specify paper width */
      case 'w':
      case 'W':
        {
          if (strlen(argv[i]) > 2)
       paper_inch = atof(argv[i++]+2);
          else
          {
       i++;
       paper_inch = atof(argv[i++]);
          }
          break;
        }
        /* specify the stretching style according to change #1 - */
        /* harry */
      case 's':
      case 'S':
        {
           stretch_mode = tolower(*(argv[i]+2));
           if ( (stretch_mode != 'n') && (stretch_mode != 'l') &&
          (stretch_mode != 'c') && (stretch_mode != 'e')
          && (stretch_mode != 'b') )
             {
                fprintf(stderr,"%s: incorrect stretch mode\n",argv[0]);
                exit(1);
             }
           if (stretch_mode == 'n') {
              if (*(argv[i]+3) == '\0') {
                 i++;
                 break;
              }
              else {
                fprintf(stderr,"%s: incorrect stretch mode\n",argv[0]);
                exit(1);
              }

           }
           else
             {
                stretch_place = tolower(*(argv[i]+3));
                if ((stretch_place != 'f') && (stretch_place != '2') &&
               (stretch_place != 'm') && (stretch_place != 'a')) {
                   fprintf(stderr,"%s: incorrect stretch place\n",argv[0]);
                   exit(1);
                }
                if (stretch_place == 'm') {
                   if (*(argv[i]+4) == '\0') {
                      if (argc > i) {
                         sscanf(argv[i+1],"%f", &stretch_amount);
                         i++;
                      }
                   }
                   else sscanf(argv[i]+4,"%f", &stretch_amount);

                   if (stretch_amount == 0.0) {
                      fprintf(stderr,"%s: incorrect stretch amount\n",argv[0]);
                      exit(1);
                   }
                   i++;
                   break;
                }
```

```
          if (*(argv[i]+4) == '\0') {
             if ((stretch_place == '2') && (stretch_mode != 'b')) {
                fprintf(stderr,
                   "%s: incorrect stretch place\n",argv[0]);
                exit(1);
             }
             i++;
             break;
          }
          if ((stretch_place != 'm') && (stretch_place != 'a')) {
             fprintf(stderr,"%s: incorrect stretch place\n",argv[0]);
             exit(1);
          }
          /* stretch_place == 'a' */

          stretch_place = tolower(*(argv[i]+4));
                         if ((stretch_place != 'd') && (stretch_place != 'l')
) {
             fprintf(stderr,"%s: incorrect stretch place\n",argv[0]);
             exit(1);
                         }
                         if (stretch_place == 'l')
       stretch_place = 'a';
                         i++;
       }
     break;
  }
  /* manual stretching control according to change #1 - harry */
case 'm':
case 'M':
  {
```

```
               if (tolower(*(argv[i]+2)) != 's') {
                  fprintf(stderr,"%s: incorrect manual stretch flag\n",argv[0]);
                  exit(1);
               }
               if (tolower(*(argv[i]+3)) == 'c')
                 msc_flag = 1;
               else if (tolower(*(argv[i]+3)) == 'l')
                 msl_flag = 1;
               else {
                  fprintf(stderr,"%s: incorrect manual stretch flag\n",argv[0]);
                  exit(1);
               }
               i++;
               break;
                 }
               default:
                 {
                   fprintf(stderr,"FFORTID: illegal argument: %s\n",argv[i]);
                   exit(1);
                   break;
                 }
             }
           }
         }
         else
         {
           printf(USAGE,argv[0]);
           exit(1);
         }
       }

```

```

/**********************************************************************/
/**********************************************************************/

  while ( (token_num=yylex()) ) /* get each token until the end-of-file */
  {


    switch (token_num)                 /* process based on token type */
    {

      case s_token:
           {
             in_size = atoi(yytext+1);
             break;
           }

      case f_token:
           {
             in_font = atoi(yytext+1);
             in_lr   = in_fontable[in_font].direction;
             strcpy(in_font_name,in_fontable[in_font].name);
             break;
           }
      case c_token:
           {
             if (new_word)
             {
               ADD_CHAR1(c_token,BEGINING,*(yytext+1));
               new_word =FALSE;
             }
```

```
                else
                {
                  ADD_CHAR1(c_token,NOT_BEGIN,*(yytext+1));
                }
                break;
              }
        case C_token:
              {
                if (new_word)
                {
                  ADD_CHAR2(C_token,BEGINING,*(yytext+1),*(yytext+2));
                  new_word = FALSE;
                }
                else
                {
                  ADD_CHAR2(C_token,NOT_BEGIN,*(yytext+1),*(yytext+2));
                }
                break;
              }
        case N_token:
              {
                if (new_word)
                {
              /* nomally N tokens have no stretch. change #1 - harry */
                  ADD_CHARN(N_token,BEGINING,*(yytext+1),*(yytext+2),*(yytext+3),NOST
RETCH);
                  new_word = FALSE;
                }
                else
                {
                  ADD_CHARN(N_token,NOT_BEGIN,*(yytext+1),*(yytext+2),*(yytext+3),NOS
TRETCH);
```

```
                  }
                  break;
                }
            case H_token:
                {
                  in_horizontal = atoi(yytext+1);
                  break;
                }
            case V_token:
                {
                  in_vertical = atoi(yytext+1);
                  break;
                }
            case h_token:
                {
                  in_horizontal = in_horizontal + atoi(yytext+1);
                  break;
                }
            case v_token:
                {
                  in_vertical = in_vertical + atoi(yytext+1);
                  break;
                }
            case hc_token:
                {
                  small_motion[0] = *(yytext);
                  small_motion[1] = *(yytext+1);
                  small_motion[2] = '\0';
                  in_horizontal = in_horizontal + atoi(small_motion);
                  if (new_word)
                  {
```

```
                ADD_CHAR1(c_token,BEGINING,*(yytext+2));
                new_word = FALSE;
              }
              else
              {
                ADD_CHAR1(c_token,NOT_BEGIN,*(yytext+2));
              }
              break;
            }
     case n_token:
            {
              if (!previous_D)
              {
                if(in_end != NULL)
                {
                  MARK_PREVIOUS_END;
                  if (lr_predom)
                    dump_line(&in_start,&in_end,RIGHT_TO_LEFT);
                  else
                  {
                    reverse_line(&in_start,&in_end,paper_width);
                    dump_line(&in_start,&in_end,LEFT_TO_RIGHT);
                  }
                }
              }
              else
                previous_D = FALSE;
              DUMP_LEX(yytext);
              break;
            }
     case w_token:
            {
              if(in_end != NULL)
                MARK_PREVIOUS_END;
              new_word = TRUE;
              break;
            }
     case p_token:
            {
              put_page_token(yytext);
              break;
            }
     case trail_token:
     case pause_token:
     case height_token:
     case slant_token:
     case include_token:
     case control_token:
            {
              DUMP_LEX(yytext);
              break;
            }
     case res_token:
            {
              DUMP_LEX(yytext);
              for(tmpyy=yytext;(*tmpyy<'0')||(*tmpyy>'9');tmpyy++);
              paper_width = paper_inch * atoi(tmpyy);
              break;
            }
     case init_token:
            {
              DUMP_LEX(yytext);
```

```
                width_init();
                **break**;
              }
        case stop_token:
              {
                printf("V%d\n",in_vertical);
                DUMP_LEX(yytext);
                break;
              }
        case newline_token:
              {
                break;
              }
        case dev_token:
              {
                device = calloc(1,strlen(yytext)-3);
                strcpy(device,yytext+4);
                DUMP_LEX(yytext);
                break;
              }
        case font_token:
              {
                font_info(yytext,&f_num,f_name);
                new_font(f_num,f_name,FONT_DIRECTION(f_num),in_fontable);
                loadfont(f_num,f_name,NULL);
                strcpy (out_fontable[f_num].name,in_fontable[f_num].name);
                out_fontable[f_num].direction = in_fontable[f_num].direction;
                DUMP_LEX(yytext);
                break;
              }
        case PR_token:
              {
                lr_predom = RIGHT_TO_LEFT;
                break;
              }
        case PL_token:
              {
                lr_predom = LEFT_TO_RIGHT;
                break;
              }

        /* stretch token added in change #1 - harry */

      case stretch_token:
        {
           token_num = yylex();
           if (token_num == newline_token)
              token_num = yylex();
           if (token_num != h_token) {
              fprintf (stderr, "FFORTID: h expected after stretch.");
              exit(1);
           }
           i = atoi(yytext+1);

           token_num = yylex();
           if (token_num != N_token) {
              fprintf (stderr, "FFORTID: N expected after stretch.");
              exit(1);
           }

           if (new_word)
              {
                 ADD_CHARN(N_token,BEGINING,*(yytext+1),
                      *(yytext+2),*(yytext+3),i);
                 new_word = FALSE;
              }
           else
              {
                 ADD_CHARN(N_token,NOT_BEGIN,*(yytext+1),
                      *(yytext+2),*(yytext+3),i);
              }
           in_horizontal += i;
           break;
        }
        case postscript_begin_token:
              {
                DUMP_LEX(yytext);
                   /* copy everything up to and including 2 lines */
                   /* containing PE\ and .\ */
                   while ( ( c = getchar() ) != EOF ) {
           putchar(c);
           if ( c == 'P' ) {
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'E') goto not_end_PS_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != '\\') goto not_end_PS_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != '\n') goto not_end_PS_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != '.') goto not_end_PS_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != '\\') goto not_end_PS_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != '\n') goto not_end_PS_yet;
              }
              goto done_flushing_PS;
           }
        not_end_PS_yet:;
                  }
               goto done;
          done_flushing_PS:
               break;
             }
        case psfig_begin_token:
             {
               printf("H%d\n",in_horizontal);
               printf("V%d\n",in_vertical);
               DUMP_LEX(yytext);
                  /* copy everything up to and including line */
                  /* containing "x X pendFig"*/
                  while ( ( c = getchar() ) != EOF ) {
           putchar(c);
           if ( c == 'x' ) {
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != ' ') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'X') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != ' ') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'p') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'e') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'n') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'd') goto not_end_psfig_yet;
              }
              if ( ( c = getchar() ) != EOF ) {
                 putchar(c);
                 if (c != 'F') goto not_end_psfig_yet;
```

```
      }
      if ( ( c = getchar() ) != EOF ) {
         putchar(c);
         if (c != 'i') goto not_end_psfig_yet;
      }
      if ( ( c = getchar() ) != EOF ) {
         putchar(c);
         if (c != 'g') goto not_end_psfig_yet;
      }
      if ( ( c = getchar() ) != EOF ) {
         putchar(c);
         if (c != '\n') goto not_end_psfig_yet;
      }
      goto done_flushing_psfig;
   }
not_end_psfig_yet:;
          }
       goto done;
  done_flushing_psfig:
       break;
     }
case D_token:
     {
       printf("H%d\n",in_horizontal);
       printf("V%d\n",in_vertical);
       printf("%s\n",yytext);
       tmpyy = yytext+1;
       if (*tmpyy == 'l' || *tmpyy == '-')
       {
         k = 0;
         while( ((*tmpyy < '0') || (*tmpyy > '9')) && (*tmpyy != '-') )
```

```
                 tmpyy++;
               while (*tmpyy != '\0')
               {
                 if(k=1-k)
                   in_horizontal += atoi(tmpyy);
                 else
                   in_vertical   += atoi(tmpyy);
                 while(((*tmpyy >= '0') && (*tmpyy <= '9')) || (*tmpyy == '-') )
                   tmpyy++;
                 while( ((*tmpyy < '0') || (*tmpyy > '9')) && (*tmpyy != '-')
                        && (*tmpyy != '\0') )
                   tmpyy++;
               }
             }
             out_horizontal = in_horizontal;
             out_vertical = in_vertical;
             previous_D = TRUE;
           }
    } /* end switch */
  }  /* end while */
  done:;
} /* end main() */
```

```
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */

/*******************************************************
********************************************************
**                                                    **
** this file contains a number of supporting routines **
**                                                    **
********************************************************
*******************************************************/

#include <stdio.h>
#include "TOKEN.h"
#include "TABLE.h"
#include "macros.h"


/************************************************************************/
/************************************************************************/


/***************************************************
****************************************************
**                                                **
** this routine adds a new font to the font table **
**                                                **
****************************************************
***************************************************/

new_font(font_number,font_name,font_direction,font_table)

```

```
  int       font_number;
  char      *font_name;
  bool      font_direction;
  TABLENTRY font_table[];

{
  strcpy(font_table[font_number].name,font_name);
  font_table[font_number].direction = font_direction;

  return;
}


/************************************************************
*************************************************************
**                                                         **
** this routine determines the width of a character token **
**                                                         **
*************************************************************
************************************************************/

font_info(font_line,font_number,font_name)

  char  *font_line;                /* LEX input token line */
  int   *font_number;              /* new font number      */
  char  *font_name[];              /* new font name        */

{

  int   i = 0;
  int   j = 0;
```

```
    char  f_num[4];

    while ((*(font_line+i) < '0') || (*(font_line+i) > '9'))
      i++;

    while ((*(font_line+i) >= '0') && (*(font_line+i) <= '9'))
    {
      f_num[j] = *(font_line+i);
      i++;
      j++;
    }
    f_num[j] = '\0';

    while (*(font_line+i) == ' ')
      i++;

    strcpy(font_name,font_line+i);
    *font_number = atoi(f_num);

    return;

  }

  /*************************************************
  **************************************************
  **                                              **
  ** this routine prints an error message and     **
  ** halts execution upon OUT OF MEMORY condition **
  **                                              **
  **************************************************
  *************************************************/
```

```

out_of_memory()
{

  fprintf(stderr,"\nFATAL ERROR: out of memory\n");
  exit(1);

}


/**************************************************************************/
/**************************************************************************/


yywrap()
{
  return 1;
}

/**************************************************************************/
/**************************************************************************/
```

```c
/* (c) Copyright 1985 Cary Buchman and Daniel M. Berry */
/* (c) Copyright 1987 Berry Computer Scientists, Ltd. */

#include <stdio.h>
#include "macros.h"

typedef struct {
   int   space_width;
   int   no_width_entries;
   char  is_special_font;
   char  font_name[10];
} Fontinfo;

#define  MAXNOFONTS  255
#define  MAXWIDENTRIES  256
#define  NOCHARSINBIGGESTFONT MAXWIDENTRIES-1   /* for no biggestfont in DESC */
#define  MAXNOCHARS  512   /* characters with two-letter or --- names */
#define  SIZECHARINDXTABLE (MAXNOCHARS + 128-32)   /* includes ascii chars,
            but not non-graphics */

Fontinfo basic_font_info[MAXNOFONTS+1];
char  font_name[MAXNOFONTS][10];

int   no_of_fonts;
int   indx_1st_spec_font;  /* index of first special font */
int   size_char_table;
int   unit_width;
int   units_per_inch;
int   no_chars_in_biggest_font = NOCHARSINBIGGESTFONT;

int   size_char_name;
```

```c
char  char_name[5*MAXNOCHARS];   /* 2 or 3 letter character names,
                              including \0 for each */
short char_table[MAXNOCHARS+1];  /* index of characters in char_name */

char  *char_indx_table[MAXNOFONTS+1];    /* fitab*/
char  *code_table[MAXNOFONTS+1];         /* codetab*/
char  *width_table[MAXNOFONTS+1];        /* widtab would be a better name */

char *connect_table[MAXNOFONTS+1];     /* connectivity table. Change */
                                    /* #1 - harry */
char *stretch_table[MAXNOFONTS+1];     /* stretchability table. Change */
                                    /* #1  - harry */

#define  FATAL 1
#define  BYTEMASK 0377

/* char  *fontdir = "/usr/lib/font"; */
char  *fontdir = "/home/harryh/usr/lib/font";

extern   char  *device; /* output device */

/**************************************************************
***************************************************************
**                                                          **
** this routine initializes the device and font width tables **
**                                                          **
***************************************************************
**************************************************************/

width_init()   /* read in font and code files, etc. */
{
	int i, j;
	char *malloc();
	char temp[60];
	FILE *descfile;
	char word[100], *ptr;

	sprintf(temp, "%s/dev%s/DESC", fontdir, device);
	if ((descfile = fopen(temp, "r")) == NULL){
	   error(FATAL, "can’t open DESC for %s\n", temp);
	}
	while (fscanf(descfile, "%s", word) != EOF) {
	   if (strcmp(word, "res") == 0) {
	      fscanf(descfile, "%d", &units_per_inch);
	   } else if (strcmp(word, "unitwidth") == 0) {
	      fscanf(descfile, "%d", &unit_width);
	   } else if (strcmp(word, "fonts") == 0) {
	      fscanf(descfile, "%d", &no_of_fonts);
	      if (no_of_fonts > MAXNOFONTS) {
	         error(FATAL,
	         "have more fonts, %d, than the maximum allowed, %d\n",
	          no_of_fonts, MAXNOFONTS);
	      }
	      /* start at 1 to leave 0 for default font */
	      for (i = 1; i <= no_of_fonts; i++) {
	         fscanf(descfile, "%s", font_name[i]);
	      }
	   } else if (strcmp(word, "biggestfont") == 0) {
	      fscanf(descfile, "%d", &no_chars_in_biggest_font);
	      if (no_chars_in_biggest_font > NOCHARSINBIGGESTFONT) {
	         error(FATAL,
	            "biggestfont of DESC too big, %d\n",
                    no_chars_in_biggest_font);
              }
           } else if (strcmp(word, "charset") == 0) {
              ptr = char_name;
              size_char_table = 0;
              while (fscanf(descfile, "%s", ptr) != EOF) {
                 if (size_char_table == MAXNOCHARS-1) {
                    error(FATAL,
                    "have more chars than the maximum allowed, %d\n",
                    MAXNOCHARS);
                 }
                 char_table[size_char_table++] = ptr - char_name;
                 while (*ptr++) /* skip to end of char name */
                    ;
              }
              size_char_name = ptr - char_name;
              char_table[size_char_table++] = 0; /* end with \0 */
           } else
           /* skip anything else */
           while (getc(descfile) != '\n');
        }
        fclose(descfile);

        width_table[0] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
        code_table[0] =(char *) malloc(MAXWIDENTRIES * sizeof(char));

        /* Change #1 - harry */

        connect_table[0] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
        stretch_table[0] =(char *) malloc(MAXWIDENTRIES *
                      sizeof(char));
```

```
   for (j = 0; j <= MAXWIDENTRIES-1; j++) {
      width_table[0][j] = 0;
      code_table[0][j] = 0;

      /* Change #1 - harry */

      connect_table[0][j] = 0;
      stretch_table[0][j] = 0;
   }
   char_indx_table[0] =(char *) malloc(SIZECHARINDXTABLE * sizeof(char));
   for (j = 0; j <= SIZECHARINDXTABLE-1; j++) {
      char_indx_table[0][j] = 0;
   }
   basic_font_info[0].space_width = 0;
   basic_font_info[0].no_width_entries = no_chars_in_biggest_font+1;
   basic_font_info[0].is_special_font = 0;
   basic_font_info[0].font_name[0] = '\0';


/* deviceprint(); /* debugging print of device */
/* fontprint(0); /*debugging print of font tables [0]*/

   for (i = 1; i <= no_of_fonts; i++) {
      width_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
      code_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));

      /* Change #1 - harry */

      connect_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
      stretch_table[i] =(char *) malloc(MAXWIDENTRIES * sizeof(char));
      for (j = 0; j <= MAXWIDENTRIES-1; j++) {
         width_table[i][j] = 0;
         code_table[i][j] = 0;

         /* Change #1 - harry */

         connect_table[i][j] = 0;
         stretch_table[i][j] = 0;
      }
      char_indx_table[i] =(char *) malloc(SIZECHARINDXTABLE *
               sizeof(char));
      for (j = 0; j <= SIZECHARINDXTABLE-1; j++) {
         char_indx_table[i][j] = 0;
      }
      getfontinfo(font_name[i],i);
/*       fontprint(i); /*debugging print of font tables [i]*/
   }
}

getfontinfo(font_name,pos)
char *font_name;
int pos;
{
   FILE *fontfile;
   int i, no_width_entries, space_width;
   char buffer[100], word[30],
      char_c[10], wid[10], asc_des[10], code[10],
           connect_c[10], stretch_c[10];  /* Change #1 - harry */


   sprintf(buffer, "%s/dev%s/%s", fontdir, device, font_name);
   if ((fontfile = fopen(buffer, "r")) == NULL){
      error(FATAL, "can’t open width table for %s\n", font_name);
   }
   while (fscanf(fontfile, "%s", word) != EOF) {
      if (word[0] == ’#’)
         while(getc(fontfile) != ’\n’);
      else if (strcmp(word, "name") == 0)
         fscanf(fontfile, "%s", basic_font_info[pos].font_name);
      else if (strcmp(word, "special") == 0)
         basic_font_info[pos].is_special_font = 1;
      else if (strcmp(word, "spacewidth") == 0) {
         fscanf(fontfile, "%d",&space_width);
         basic_font_info[pos].space_width = space_width;
         if (space_width == 0) {
            /* Rounding error fix. Change #1 - harry */

            width_table[pos][0] = units_per_inch * unit_width / 72.0 / 3.0;
         } else {
            width_table[pos][0] = space_width;
         }
      } else if (strcmp(word, "charset") == 0) {
         while(getc(fontfile) != ’\n’);
         no_width_entries = 0;
         /* widths are origin 1 so char_indx_table entry of
         0 can mean not there */
         while (fgets(buffer, 100, fontfile) != NULL) {

            /* Change #1 - harry */

            sscanf(buffer, "%s %s %s %s %s %s",
               char_c, wid, asc_des, code,
                   connect_c, stretch_c);
```

```

                 if (wid[0] != '"') { /* not a ditto */
                    no_width_entries++;
                    width_table[pos][no_width_entries] =
                       atoi(wid);

                    if (code[0] == '0')
                       sscanf(code, "%o", &i);
                    else
                       sscanf(code, "%d", &i);
                    code_table[pos][no_width_entries] = i;
                 }
                 /* otherwise a synonym for previous character,
                 so leave previous values intact */
                 if (strlen(char_c) == 1)   /* it's ascii */
                    char_indx_table[pos][char_c[0]-32] =
                       no_width_entries;
                       /* char_indx_table origin
                       omits non-graphics */
                 else if (strcmp(char_c, "---") != 0) {
                       /* it has a 2-char name */
                    for (i = 0; i <= size_char_table; i++)
                       if (strcmp(
                           &char_name[char_table[i]],
                           char_c) == 0) {
                          char_indx_table[pos][i+128-32] =
                             no_width_entries;
                             /* starts after the ascii */
                          break;
                       }
                    if (i >= size_char_table)
                       error(FATAL,
                           "font %s: %s not in charset\n",
                           font_name, char_c);
                 }
                 else {
                    /* Change #1 - harry */

                    connect_table[pos][no_width_entries] = connect_c[0];
                    stretch_table[pos][no_width_entries] = stretch_c[0];
                 }
              }
              if (no_width_entries > MAXWIDENTRIES) {
                 error(FATAL,
                    "font has %d characters, too big\n",
                    no_width_entries);
              }
              basic_font_info[pos].no_width_entries =
                 no_chars_in_biggest_font+1;
           }
             else while(getc(fontfile) != '\n');
        }
        fclose(fontfile);
     }

     /********************************************************
     *********************************************************
     **                                                    **
     ** this routine prints the specified font width table **
     **                                                    **
     *********************************************************
     ********************************************************/
```

```

fontprint(i)   /* debugging print of font i (0,...) */
{
   int jj, kk, nn;

   printf("font %d:\n", i);
   nn = basic_font_info[i].no_width_entries;

   printf("base=0xxxxxx, nchars=%d, spec=%d, name=%s, width_table=0xxxxxx, char_indx_table=0xxxxxx, code_table=0xxxxxx\n",
      nn, basic_font_info[i].is_special_font, basic_font_info[i].font_name);

   printf("\nwidths:\n");
   for (jj=0; jj <= nn; jj++) {
      printf(" %2d", width_table[i][jj] & BYTEMASK);
      if (jj % 20 == 19) printf("\n");
   }

   printf("\nchar_indx_table:\n");
   for (jj=0; jj < size_char_table + 128-32; jj++) {
      printf(" %2d", char_indx_table[i][jj] & BYTEMASK);
      if (jj % 20 == 19) printf("\n");
   }

   printf("\ncode_table:\n");
   for (jj=0; jj <= nn; jj++) {
      printf(" %2d", code_table[i][jj] & BYTEMASK);
      if (jj % 20 == 19) printf("\n");
   }

   printf("\n");
```

```
}

/*******************************************************
********************************************************
**                                                   **
** this routine prints the device table **
**                                                   **
********************************************************
*******************************************************/

deviceprint()  /* debugging print of device */
{
   int j;
   int jj;

   printf("device:\n");

   printf("res=%d nfonts=%d nchtab=%d unitwidth=%d lchname=%d",
      units_per_inch, no_of_fonts, size_char_table, unit_width,
      size_char_name);

   printf("\nchtab:\n");
   for (jj=0; jj <= size_char_table-1; jj++) {
      printf(" %2d", char_table[jj]);
      if (jj % 20 == 19) printf("\n");
   }

   printf("\nchname:\n");
   for (jj=0; jj <= size_char_table-1; jj++) {
      printf(" %s", &char_name[char_table[jj]]);
      if (jj % 20 == 19) printf("\n");
```

```
    }

    printf("\n");
}

/*******************************************************
********************************************************
**                                                    **
** this routine loads the specified font width table **
**                                                    **
********************************************************
*******************************************************/


loadfont(n, s, s1)   /* load font info for font s on position n (0...) */
int n;
char *s, *s1;
{
/*
   char temp[60];
   int fin, nw, norig;


   if (n < 0 || n > MAXNOFONTS)
      error(FATAL, "illegal fp command %d %s", n, s);
   if (strcmp(s, fontbase[n]->namefont) == 0)
      return;
   if (s1 == NULL || s1[0] == '\0')
      sprintf(temp, "%s/dev%s/%s.out", fontdir, device, s);
   else
      sprintf(temp, "%s/%s.out", s1, s);
   if ((fin = open(temp, 0)) < 0)
      error(FATAL, "can't open font table %s", temp);
   norig = fontbase[n]->nwfont & BYTEMASK;
   read(fin, fontbase[n], 3 * norig + size_char_table+128-32 + sizeof(struct Font)
);
   if ((fontbase[n]->nwfont & BYTEMASK) > norig)
      error(FATAL, "Font %s too big for position %d\n", s, n);
   close(fin);
   nw = fontbase[n]->nwfont & BYTEMASK;
   o_width_table[n] = (char *) fontbase[n] + sizeof(struct Font);
   o_char_indx_table[n] = (char *) o_width_table[n] + 3 * nw;
   fontbase[n]->nwfont = norig;
*/
}


error(f, s, a1, a2, a3, a4, a5, a6, a7) {
   fprintf(stderr, "ffortid: ");
   fprintf(stderr, s, a1, a2, a3, a4, a5, a6, a7);
   fprintf(stderr, "\n");
   if (f)
          exit(1);
}

/************************************************************************
*************************************************************************
**                                                                     **
** this routine determines the width of the specified funny character **
**                                                                     **
*************************************************************************
************************************************************************/
```

```c


width2(s,in_size,in_font)  /* s is a funny char name */
char *s;
int  in_size;     /* Johny */
int    in_font;
{
   int i;

   for (i = 0; i < size_char_table; i++)
     if (strcmp(&char_name[char_table[i]], s) == 0)
       break;
   if (i < size_char_table)
         return(width1(i + 128,in_size,in_font));
        else
         return(width1(0,in_size,in_font));
}


/* All the following functions were changed or added in change #1 - */
/* harry */

/*******************************************************************
********************************************************************
**                                                               **
** this routine determines the width of the specified character **
**                                                               **
** in_size is passed as a parameter as this procedure is used   **
** to calculate the filler width for each word of the line.     **
** it is necessary because the in_size at the start and end of  **
** the line may be different. Johny                    **
**                                                  **
*******************************************************************
******************************************************************/


width1(c,in_size,in_font)  /* output char c */
int c;
int in_size;
int in_font;
{
   char *pw;
   register char *p;
   register int i, k;
   int j, w, width;

   c -= 32;
   if (c <= 0)
      return(widthToGoobies(width_table[in_font][0],in_size));
   k = in_font;
   i = char_indx_table[in_font][c] & BYTEMASK;
   if (i != 0) {  /* it's on this font */
      pw = width_table[in_font];
   } else if (indx_1st_spec_font > 0) { /* on special (we hope) */
      for (k=indx_1st_spec_font, j=0;
            j < no_of_fonts; j++, k = k % no_of_fonts + 1)
         if ((i = char_indx_table[k][c] & BYTEMASK) != 0) {
            pw = width_table[k];
            break;
         }
   }
   if (i == 0 || j == no_of_fonts) {
        return(widthToGoobies(width_table[in_font][0],in_size));
     }
     width = pw[i] & BYTEMASK;

     width = widthToGoobies(width, in_size);
          return(width);
}



/******************************************************************
*******************************************************************
**                                                              **
** this routine determines the width of the character whose     **
** code is n, i.e. specified Nn                                 **
**                                                              **
*******************************************************************
******************************************************************/


widthn(pn,in_size,in_font) /* output char with abs code *pn */
char *pn;
int in_size;
int in_font;
{
   char *pw;
   register int i;
   int n,width;

   sscanf(pn, "%d", &n); /*get the string *pn and convert to integer*/

   i= abscw (n,in_font);
   pw=width_table[in_font];
   width = pw[i] & BYTEMASK;

   /* rounding error fix. Change #1 - harry */

   width = widthToGoobies(width, in_size);
   return(width);
}


/*****************************************************************
******************************************************************
**                                                              **
** this routine determines the index of abs char n in           **
**  width_table[]                                               **
**                                                              **
******************************************************************
*****************************************************************/

abscw(n,in_font)
int n;
int in_font;
{
   register int i, ncf;

   ncf= basic_font_info[in_font].no_width_entries & BYTEMASK;
   for (i=0; i< ncf; i++)
      if ((unsigned char)code_table[in_font][i] == n)
      /* a bug fix for the \N'xxx' to work, when xxx > 128 */
         return i;
   return 0;
}


/*****************************************************************
******************************************************************
**                                                              **
** this routine converts width table value to goobies for a     **
**   certain point size                                         **
**                                                              **
******************************************************************
*****************************************************************/

int widthToGoobies(width, point_size)
int width;
int point_size;
{
   return (int) (((float) width * point_size / unit_width) + 0.5) ;
}

/*****************************************************************
******************************************************************
**                                                              **
** this routine returns the connection properties of abs char n **
**                                                              **
******************************************************************
*****************************************************************/

char connect_properties(n,in_font)
int n;
int in_font;
```

```
{
   return (connect_table[in_font][abscw(n,in_font)]);
}


/*****************************************************************
******************************************************************
**                                                              **
** this routine returns whether abs char n is a connect         **
**     previous letter                                          **
**                                                              **
******************************************************************
*****************************************************************/

int connectable(n,in_font)
int n;
int in_font;
{
   char c;

   c = connect_properties(n,in_font);
   return ((c == CONNECTPREVIOUS) || (c == CONNECTBOTH));
}


/*****************************************************************
******************************************************************
**                                                              **
** this routine returns whether abs char n is a stretchable     **
**   letter                                                     **
**                                                              **
```

```
*****************************************************************
****************************************************************/

int stretchable(n,in_font)
int n;
int in_font;
{
   return (stretch_table[in_font][abscw(n,in_font)] == STRETCHABLE);
}
```