

PotatOS

Generated by Doxygen 1.8.11

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	ALIAS Struct Reference	5
3.1.1	Detailed Description	5
3.2	COMMAND Struct Reference	5
3.2.1	Detailed Description	6
3.3	control_sequence Struct Reference	6
3.3.1	Detailed Description	6
3.4	HELP_PAGES Struct Reference	7
3.4.1	Detailed Description	7
3.5	node Struct Reference	7
3.5.1	Detailed Description	8
3.6	pcb Struct Reference	8
3.6.1	Detailed Description	8
3.7	queue Struct Reference	9
3.7.1	Detailed Description	9
3.8	time Struct Reference	9
3.8.1	Detailed Description	10

4 File Documentation	11
4.1 command_handler.h File Reference	11
4.1.1 Detailed Description	11
4.2 commands.h File Reference	11
4.2.1 Detailed Description	12
4.2.2 Function Documentation	12
4.2.2.1 cmd_blockPCB(char *params)	12
4.2.2.2 cmd_clear(char *params)	12
4.2.2.3 cmd_create_pcb(char *params)	13
4.2.2.4 cmd_date(char *params)	13
4.2.2.5 cmd_delete_pcb(char *params)	13
4.2.2.6 cmd_help(char *params)	13
4.2.2.7 cmd_resume(char *params)	14
4.2.2.8 cmd_set_priority_pcb(char *params)	14
4.2.2.9 cmd_shutdown(char *params)	14
4.2.2.10 cmd_suspend(char *params)	14
4.2.2.11 cmd_time(char *params)	15
4.2.2.12 cmd_unblock_pcb(char *params)	15
4.2.2.13 cmd_version(char *params)	15
4.3 commandUtils.h File Reference	16
4.3.1 Detailed Description	17
4.3.2 Macro Definition Documentation	17
4.3.2.1 A_FLAG	17
4.3.2.2 alphanum	17
4.3.2.3 B_FLAG	17
4.3.2.4 C_FLAG	17
4.3.2.5 CMDSIZE	18
4.3.2.6 D_FLAG	18
4.3.2.7 E_FLAG	18
4.3.2.8 F_FLAG	18

4.3.2.9	G_FLAG	18
4.3.2.10	H_FLAG	18
4.3.2.11	I_FLAG	18
4.3.2.12	J_FLAG	18
4.3.2.13	K_FLAG	18
4.3.2.14	L_FLAG	18
4.3.2.15	M_FLAG	19
4.3.2.16	N_FLAG	19
4.3.2.17	NO_FLAG	19
4.3.2.18	O_FLAG	19
4.3.2.19	P_FLAG	19
4.3.2.20	Q_FLAG	19
4.3.2.21	R_FLAG	19
4.3.2.22	S_FLAG	19
4.3.2.23	T_FLAG	19
4.3.2.24	U_FLAG	19
4.3.2.25	V_FLAG	20
4.3.2.26	W_FLAG	20
4.3.2.27	X_FLAG	20
4.3.2.28	Y_FLAG	20
4.3.2.29	Z_FLAG	20
4.3.3	Function Documentation	20
4.3.3.1	get_pvalue(char c)	20
4.3.3.2	set_flags(char *paramstr, int *flag, int num_aliases,...)	20
4.3.3.3	set_flags_search_alias(char *alias, int num_aliases, ALIAS aliases[])	21
4.3.4	Variable Documentation	21
4.3.4.1	gparamstr	21
4.4	pcb_constants.h File Reference	21
4.4.1	Detailed Description	22
4.4.2	Typedef Documentation	23

4.4.2.1	node_t	23
4.5	pcb_queue.h File Reference	23
4.5.1	Detailed Description	24
4.5.2	Function Documentation	24
4.5.2.1	construct_queue()	24
4.5.2.2	dequeue(queue_t *queue)	24
4.5.2.3	destruct_queue(queue_t *queue)	24
4.5.2.4	enqueue(queue_t *que, pcb_t *data)	26
4.5.2.5	priority_enqueue(queue_t *cue, pcb_t *data)	26
4.6	pcb_utils.h File Reference	26
4.6.1	Detailed Description	28
4.6.2	Function Documentation	28
4.6.2.1	allocate_pcb()	28
4.6.2.2	find_pcb(char *pname)	28
4.6.2.3	free_pcb(pcb_t *)	28
4.6.2.4	get_blocked_queue()	28
4.6.2.5	get_process_class_string(PROCESS_CLASS process_class)	28
4.6.2.6	get_process_state_string(PROCESS_STATE process_state)	29
4.6.2.7	get_ready_queue()	29
4.6.2.8	get_suspended_blocked_queue()	29
4.6.2.9	get_suspended_ready_queue()	29
4.6.2.10	print_pcb_info(const pcb_t *pcb)	30
4.6.2.11	remove_pcb(char *pname)	30
4.6.2.12	setup_pcb(char *, PROCESS_CLASS, int priority)	30
4.7	pcb_wrangler.h File Reference	30
4.7.1	Detailed Description	30
4.8	poll_input.c File Reference	31
4.8.1	Detailed Description	32
4.8.2	Function Documentation	32
4.8.2.1	get_key()	32

4.8.2.2	input_available()	32
4.8.2.3	memcpy(char *destination, const char *source, int n)	32
4.8.2.4	move_cursor(int n)	33
4.8.2.5	poll_input(char *buffer, int *length)	33
4.8.2.6	print_after_cursor(const char *str)	33
4.8.2.7	wait_for_input(int timeout)	33
4.8.3	Variable Documentation	34
4.8.3.1	control_sequences	34
4.8.3.2	TOLERANCE	34
4.9	poll_input.h File Reference	34
4.9.1	Detailed Description	35
4.9.2	Typedef Documentation	35
4.9.2.1	ControlSequence	35
4.9.3	Function Documentation	35
4.9.3.1	poll_input(char *buffer, int *length)	36
4.10	splash.h File Reference	36
4.10.1	Detailed Description	36
4.11	stdio.c File Reference	37
4.11.1	Detailed Description	37
4.11.2	Function Documentation	37
4.11.2.1	printf(char *form,...)	37
4.11.2.2	puts(char *buff)	38
4.12	stdio.h File Reference	38
4.12.1	Detailed Description	39
4.12.2	Function Documentation	39
4.12.2.1	printf(char *form,...)	39
4.12.2.2	puts(char *buffer)	39
4.13	string.c File Reference	39
4.13.1	Detailed Description	41
4.13.2	Function Documentation	41

4.13.2.1	<code>isdigit(char c)</code>	41
4.13.2.2	<code>itoa(int num, char *str, int base)</code>	41
4.13.2.3	<code>reverse(char *str, int j)</code>	41
4.13.2.4	<code>sprintf(char *buffer, char *format,...)</code>	42
4.13.2.5	<code>sprintf_internal(char *buffer, char *format, va_list valist)</code>	42
4.13.2.6	<code>sprintf_pad_helper(char *buffer, char pad, int fNum, int n, BYTE doAction)</code>	43
4.13.2.7	<code>tolower(char c)</code>	44
4.13.2.8	<code>toupper(char c)</code>	44
4.13.2.9	<code>trim(char *str)</code>	44
4.14	<code>string.h</code> File Reference	45
4.14.1	Detailed Description	46
4.14.2	Function Documentation	46
4.14.2.1	<code>isdigit(char c)</code>	46
4.14.2.2	<code>itoa(int num, char *str, int base)</code>	46
4.14.2.3	<code>reverse(char *str, int j)</code>	47
4.14.2.4	<code>sprintf(char *buffer, char *format,...)</code>	47
4.14.2.5	<code>sprintf_internal(char *buffer, char *format, va_list valist)</code>	47
4.14.2.6	<code>tolower(char c)</code>	48
4.14.2.7	<code>toupper(char c)</code>	48
4.14.2.8	<code>trim(char *str)</code>	48
4.15	<code>time.h</code> File Reference	48
4.15.1	Detailed Description	49
4.15.2	Function Documentation	49
4.15.2.1	<code>bcd_to_decimal(int bcd)</code>	49
4.15.2.2	<code>format_time(char *dest, time_h *t)</code>	50
4.15.2.3	<code>get_current_time()</code>	50
4.15.2.4	<code>set_current_time(time_h time)</code>	50
4.16	<code>utility.c</code> File Reference	51
4.16.1	Detailed Description	51
4.16.2	Function Documentation	51
4.16.2.1	<code>isnullorspace(char test)</code>	51
4.17	<code>utility.h</code> File Reference	52
4.17.1	Detailed Description	52
4.17.2	Function Documentation	52
4.17.2.1	<code>isnullorspace(char test)</code>	52

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

ALIAS	A struct to hold command aliases	5
COMMAND	A struct to hold commands	5
control_sequence	A struct to hold key mappings	6
HELP_PAGES	A struct to hold help outputs	7
node	One element within the pcb queue	7
pcb	Struct that contains all information related to a pcb	8
queue	Contains all the data needed to use/modify a queue	9
time	A struct to all the time and date elements	9

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

command_handler.h	The header file for the command handler for the Operating System	11
commands.h	The header file for commands.c	11
commandUtils.h	Utilites that apply to all command files	16
pcb_constants.h	Contains all shared resources amongst all PCBs	21
pcb_queue.h	File to hold all queue functions	23
pcb_utils.h	Utility functions for all PCBs	26
pcb_wrangler.h	Initiates the creation of all queues	30
poll_input.c	The polling input file that allows user input	31
poll_input.h	The header file for the polling input	34
splash.h	File to hold the splash screen	36
stdio.c	Holds all implementation of standard I/O functions	37
stdio.h	Holds all prototypes of standard I/O functions	38
string.c	Holds all utility functions used to modify strings	39
string.h	Holds all utility prototypes used to modify strings	45
time.h	The header file for the date and time functions	48
utility.c	Holds utility function implementations for this project	51
utility.h	Holds utility function prototypes for this project	52

Chapter 3

Data Structure Documentation

3.1 ALIAS Struct Reference

A struct to hold command aliases.

```
#include <commandUtils.h>
```

Data Fields

- char **c**
- char * **val**

3.1.1 Detailed Description

A struct to hold command aliases.

The [ALIAS](#) Struct is a custom struct that is designed to hold aliases for commands

Parameters

<i>c</i>	A string that will hold the initial command name
<i>val</i>	A string pointer that will point to the original command name

The documentation for this struct was generated from the following file:

- [commandUtils.h](#)

3.2 COMMAND Struct Reference

A struct to hold commands.

Data Fields

- char * **str**
- int(* **func**)(char *)

3.2.1 Detailed Description

A struct to hold commands.

The [COMMAND](#) Struct is a custom struct that is designed to hold custom commands

Parameters

<i>str</i>	A string type to hold the name of the command
<i>CommandPointer</i>	A pointer to a command so that we can pass commands

The documentation for this struct was generated from the following file:

- [command_handler.c](#)

3.3 control_sequence Struct Reference

A struct to hold key mappings.

```
#include <poll_input.h>
```

Data Fields

- char **code** [8]
- int **id**

3.3.1 Detailed Description

A struct to hold key mappings.

The [control_sequence](#) Struct is a custom struct that is designed to hold mappings between control sequence codes used to encode arrow keys. It also holds other special buttons.

Parameters

<i>code</i>	The special keyboard code name
<i>id</i>	The keyboard code value

The documentation for this struct was generated from the following file:

- [poll_input.h](#)

3.4 HELP_PAGES Struct Reference

A struct to hold help outputs.

Data Fields

- char * **command_name**
- char * **command_help_page**

3.4.1 Detailed Description

A struct to hold help outputs.

The [COMMAND](#) Struct is a custom struct that is designed to hold custom commands

Parameters

<i>str</i>	A string type to hold the name of the command
<i>command_help_page</i>	A string that holds the actual help page

The documentation for this struct was generated from the following file:

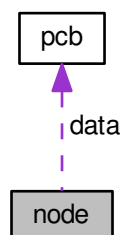
- cmdHelp.c

3.5 node Struct Reference

One element within the pcb queue.

```
#include <pcb_constants.h>
```

Collaboration diagram for node:



Data Fields

- [pcb_t](#) * **data**
- void * **next**
- void * **prev**

3.5.1 Detailed Description

One element within the pcb queue.

This allows us to abbreviate code elsewhere... probably

The documentation for this struct was generated from the following file:

- [pcb_constants.h](#)

3.6 pcb Struct Reference

Struct that contains all information related to a pcb.

```
#include <pcb_constants.h>
```

Data Fields

- char * **process_name**
- unsigned int **process_class**
- unsigned int **priority**
- unsigned int **state**
- char **stack** [2048]

3.6.1 Detailed Description

Struct that contains all information related to a pcb.

The documentation for this struct was generated from the following file:

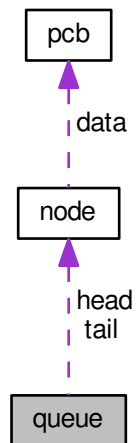
- [pcb_constants.h](#)

3.7 queue Struct Reference

Contains all the data needed to use/modify a queue.

```
#include <pcb_constants.h>
```

Collaboration diagram for queue:



Data Fields

- int **size**
- struct `node` * **head**
- struct `node` * **tail**

3.7.1 Detailed Description

Contains all the data needed to use/modify a queue.

The documentation for this struct was generated from the following file:

- [pcb_constants.h](#)

3.8 time Struct Reference

A struct to all the time and date elements.

```
#include <time.h>
```

Data Fields

- int **seconds**
- int **minutes**
- int **hours**
- int **day_of_month**
- int **month**
- int **year**

3.8.1 Detailed Description

A struct to all the time and date elements.

The time Struct is a custom struct that is designed to hold all the elements necessary for time and date.

The documentation for this struct was generated from the following file:

- [time.h](#)

Chapter 4

File Documentation

4.1 `command_handler.h` File Reference

The header file for the command handler for the Operating System.

Functions

- int `command_handler` ()
Entry point for the command handler.

4.1.1 Detailed Description

The header file for the command handler for the Operating System.

4.2 `commands.h` File Reference

The header file for `commands.c`.

Functions

- int `cmd_help` (char *params)
The help command will show a page to assist users with commands.
- int `cmd_version` (char *params)
The version command will show the version information.
- int `cmd_shutdown` (char *params)
shutdown the PotatOS
- int `cmd_date` (char *params)
The date command will do one of two things. Show the current system date Set a new system date.
- int `cmd_time` (char *params)
The time command will do one of two things. Show the current system time Set a new system time.
- int `cmd_test` (char *params)

- int `cmd_clear` (char *params)
clears the screen and sets the pointer at home
- int `cmd_create_pcb` (char *params)
Create a new pcb.
- int `cmd_unblock_pcb` (char *params)
Unblock a pcb.
- int `cmd_blockPCB` (char *params)
command to block PCB by name
- int `cmd_resume` (char *params)
Resume PCB command.
- int `cmd_suspend` (char *params)
Suspend PCB command.
- int `cmd_show_pcb` (char *params)
Show PCB command.
- int `cmd_show_all_pcb`s (char *params)
Show all PCBs command.
- int `cmd_show_ready_pcb`s (char *params)
Show ready PCBs command.
- int `cmd_show_blocked_pcb`s (char *params)
Show blocked PCBs command.
- int `cmd_delete_pcb` (char *params)
command to delete PCB by name
- int `cmd_set_priority_pcb` (char *params)
command to set PCB priority

4.2.1 Detailed Description

The header file for commands.c.

4.2.2 Function Documentation

4.2.2.1 int cmd_blockPCB (char * params)

command to block PCB by name

Returns

Success or Failure

4.2.2.2 int cmd_clear (char * params)

clears the screen and sets the pointer at home

Parameters

<i>params</i>	param string typed by user
---------------	----------------------------

Returns

SUCCESS or FAILURE

4.2.2.3 int cmd_create_pcb (char * *params*)

Create a new pcb.

Parameters

<i>params</i>	string passed from command handler
---------------	------------------------------------

Returns

SUCCESS or FAILURE

4.2.2.4 int cmd_date (char * *params*)

The date command will do one of two things. Show the current system date Set a new system date.

The date command can be used to query the systems RTC to display the current date. It can also be used to set the systems RTC to a desired date. There is code to check for illegal dates such as Feb 30 on a non leap year.

Parameters

<i>params</i>	param string typed by user
---------------	----------------------------

Returns

The current system date

Warning

The RTC only allows dates between 1700-2999

4.2.2.5 int cmd_delete_pcb (char * *params*)

command to delete PCB by name

Returns

Success if the PCB was removed, failure for anything else

4.2.2.6 int cmd_help (char * *params*)

The help command will show a page to assist users with commands.

The help command can be called to do one of two things List all the commands that have help pages Request a help page for a certain command

Parameters

<i>params</i>	param string typed by user
---------------	----------------------------

Returns

A help page

4.2.2.7 int cmd_resume (char * *params*)

Resume PCB command.

Returns

SUCCESS or FAILURE

4.2.2.8 int cmd_set_priority_pcb (char * *params*)

command to set PCB priority

Returns

Success if the PCB priority was updated, failure for anything else

4.2.2.9 int cmd_shutdown (char * *params*)

shutdown the PotatOS

Parameters

<i>params</i>	string passed from command handler
---------------	------------------------------------

Returns

SUCCESS or FAILURE

4.2.2.10 int cmd_suspend (char * *params*)

Suspend PCB command.

Returns

SUCCESS or FAILURE

4.2.2.11 int cmd_time (char * *params*)

The time command will do one of two things. Show the current system time Set a new system time.

The time command can be used to query the systems RTC to display the current time. It can also be used to set the systems RTC to a desired time. There is code to check for illegal times.

Parameters

<i>params</i>	param string typed by user
---------------	----------------------------

Returns

The current system time

Note

The time is kept in 24 hour time

4.2.2.12 int cmd_unblock_pcb (char * *params*)

Unblock a pcb.

Parameters

<i>params</i>	string passed from command handler
---------------	------------------------------------

Returns

SUCCESS or FAILURE

4.2.2.13 int cmd_version (char * *params*)

The version command will show the version information.

The version command can be called to display the version information. The shortened return will just show the short version. The long return will include the current module, the version, and the contributing developers

Parameters

<i>params</i>	param string typed by user
---------------	----------------------------

Returns

A version page

4.3 commandUtils.h File Reference

Utilites that apply to all command files.

Data Structures

- struct [ALIAS](#)
A struct to hold command aliases.

Macros

- #define [CMD_SIZE](#) 100
The command input buffer.
 - #define [SUCCESS](#) 1
Macro to return a 0 on success.
 - #define [FAILURE](#) 0
Macro to return a -1 on failure.
 - #define [MAXPARAMCOUNT](#) 10
The maximum parameters allowed per command.
 - #define [A_FLAG](#) (1 << 0)
 - #define [B_FLAG](#) (1 << 1)
 - #define [C_FLAG](#) (1 << 2)
 - #define [D_FLAG](#) (1 << 3)
 - #define [E_FLAG](#) (1 << 4)
 - #define [F_FLAG](#) (1 << 5)
 - #define [G_FLAG](#) (1 << 6)
 - #define [H_FLAG](#) (1 << 7)
 - #define [I_FLAG](#) (1 << 8)
 - #define [J_FLAG](#) (1 << 9)
 - #define [K_FLAG](#) (1 << 10)
 - #define [L_FLAG](#) (1 << 11)
 - #define [M_FLAG](#) (1 << 12)
 - #define [N_FLAG](#) (1 << 13)
 - #define [O_FLAG](#) (1 << 14)
 - #define [P_FLAG](#) (1 << 15)
 - #define [Q_FLAG](#) (1 << 16)
 - #define [R_FLAG](#) (1 << 17)
 - #define [S_FLAG](#) (1 << 18)
 - #define [T_FLAG](#) (1 << 19)
 - #define [U_FLAG](#) (1 << 20)
 - #define [V_FLAG](#) (1 << 21)
 - #define [W_FLAG](#) (1 << 22)
 - #define [Y_FLAG](#) (1 << 23)
 - #define [X_FLAG](#) (1 << 24)
 - #define [Z_FLAG](#) (1 << 25)
 - #define [NO_FLAG](#) (1<<26)
 - #define [alphanum](#)(c) (('a' <= c && c <= 'z') ? c - 'a' : c - 'A')
- A helper macro that will take a letter and return its integer equivalent.*

Functions

- int `set_flags` (char *paramstr, int *flag, int num_aliases,...)
Sets flags based on param string, flags and num aliases.
- char * `get_pvalue` (char c)
Gets value of specific flag.
- char `set_flags_search_alias` (char *alias, int num_aliases, `ALIAS` aliases[])
Used as a helper function for set_flags.

Variables

- char `gparamstr` [`CMDSIZE`]
A string to hold the command input up to the max command size.
- char * `gparams` [27]
Will hold all the string pointers.

4.3.1 Detailed Description

Utilites that apply to all command files.

4.3.2 Macro Definition Documentation

4.3.2.1 `#define A_FLAG (1 << 0)`

`cmd_help` flags A flag binary bit shift macro

4.3.2.2 `#define alphanum(c) (('a' <= c && c <= 'z') ? c - 'a' : c - 'A')`

A helper macro that will take a letter and return its integer equivalent.

This is a helper macro that is used in `set_flags` and `get_gparams`. It takes in character and return the integer equivalent of that character.

Parameters

<code>c</code>	The character to be returned as an int
----------------	--

4.3.2.3 `#define B_FLAG (1 << 1)`

B flag binary bit shift macro

4.3.2.4 `#define C_FLAG (1 << 2)`

C flag binary bit shift macro

4.3.2.5 `#define CMDSIZE 100`

The command input buffer.

This a macro to store the command input buffer. Here we can change the amount of characters we allow to be entered into the command handler at once. We currently allow 100 characters.

4.3.2.6 `#define D_FLAG (1 << 3)`

D flag binary bit shift macro

4.3.2.7 `#define E_FLAG (1 << 4)`

E flag binary bit shift macro

4.3.2.8 `#define F_FLAG (1 << 5)`

F flag binary bit shift macro

4.3.2.9 `#define G_FLAG (1 << 6)`

G flag binary bit shift macro

4.3.2.10 `#define H_FLAG (1 << 7)`

H flag binary bit shift macro

4.3.2.11 `#define I_FLAG (1 << 8)`

I flag binary bit shift macro

4.3.2.12 `#define J_FLAG (1 << 9)`

J flag binary bit shift macro

4.3.2.13 `#define K_FLAG (1 << 10)`

K flag binary bit shift macro

4.3.2.14 `#define L_FLAG (1 << 11)`

L flag binary bit shift macro

4.3.2.15 #define M_FLAG (1 << 12)

M flag binary bit shift macro

4.3.2.16 #define N_FLAG (1 << 13)

N flag binary bit shift macro

4.3.2.17 #define NO_FLAG (1 << 26)

NO flag binary bit shift macro

4.3.2.18 #define O_FLAG (1 << 14)

O flag binary bit shift macro

4.3.2.19 #define P_FLAG (1 << 15)

P flag binary bit shift macro

4.3.2.20 #define Q_FLAG (1 << 16)

Q flag binary bit shift macro

4.3.2.21 #define R_FLAG (1 << 17)

R flag binary bit shift macro

4.3.2.22 #define S_FLAG (1 << 18)

S flag binary bit shift macro

4.3.2.23 #define T_FLAG (1 << 19)

T flag binary bit shift macro

4.3.2.24 #define U_FLAG (1 << 20)

U flag binary bit shift macro

4.3.2.25 `#define V_FLAG (1 << 21)`

V flag binary bit shift macro

4.3.2.26 `#define W_FLAG (1 << 22)`

W flag binary bit shift macro

4.3.2.27 `#define X_FLAG (1 << 24)`

X flag binary bit shift macro

4.3.2.28 `#define Y_FLAG (1 << 23)`

Y flag binary bit shift macro

4.3.2.29 `#define Z_FLAG (1 << 25)`

Z flag binary bit shift macro

4.3.3 Function Documentation

4.3.3.1 `char* get_pvalue (char c)`

Gets value of specific flag.

Usage: `get_pvalue('a');`

Parameters

<code>c</code>	character of flag to get the value from
----------------	---

Returns

value after the flag specified

4.3.3.2 `int set_flags (char * paramstr, int * flag, int num_aliases, ...)`

Sets flags based on param string, flags and num aliases.

Usage: `set_flags(paramstr,&flag,5, 'a',"alpha", 'b',"bravo", 'f',"foxtrot", 'g',"golf", 'r',"whiskey")`

Parameters

<i>paramstr</i>	string that each command gets. Typed by the user
<i>flag</i>	pointer to integer flag
<i>num_aliases</i>	number of aliases specified

Returns

success or failure

Note

num_aliases must be the exact number of parameters. In the example, 5

4.3.3.3 char set_flags_search_alias (char * alias, int num_aliases, ALIAS aliases[])

Used as a helper function for set_flags.

Parameters

<i>alias</i>	alias to search for in aliases
<i>num_aliases</i>	number of aliases in aliases
<i>aliases</i>	array of ALIASes to search through

Returns

character of flag that it found

4.3.4 Variable Documentation

4.3.4.1 char gparamstr[CMD_SIZE]

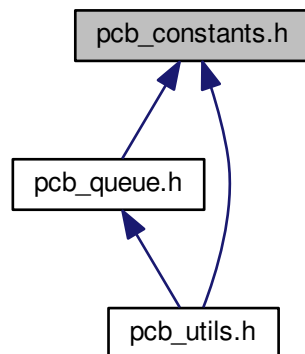
A string to hold the command input up to the max command size.

A flag binary bit shift macro

4.4 pcb_constants.h File Reference

Contains all shared resources amongst all PCBs.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct `pcb`
Struct that contains all information related to a pcb.
- struct `node`
One element within the pcb queue.
- struct `queue`
Contains all the data needed to use/modify a queue.

Typedefs

- typedef struct `pcb` `pcb_t`
Struct that contains all information related to a pcb.
- typedef struct `node` `node_t`
One element within the pcb queue.
- typedef struct `queue` `queue_t`
Contains all the data needed to use/modify a queue.

Enumerations

- enum `PROCESS_CLASS` { `SYSTEM`, `APPLICATION` }
Contains all possible process classes.
- enum `PROCESS_STATE` { `RUNNING`, `READY`, `BLOCKED`, `SUSPENDED_READY`, `SUSPENDED_BLOCKED` }
Contains all possible process states.

4.4.1 Detailed Description

Contains all shared resources amongst all PCBs.

4.4.2 Typedef Documentation

4.4.2.1 typedef struct node node_t

One element within the pcb queue.

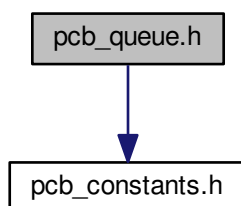
This allows us to abbreviate code elsewhere... probably

4.5 pcb_queue.h File Reference

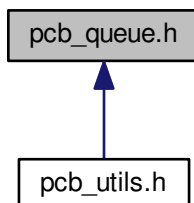
File to hold all queue functions.

```
#include "pcb_constants.h"
```

Include dependency graph for pcb_queue.h:



This graph shows which files directly or indirectly include this file:



Functions

- void `enqueue (queue_t *que, pcb_t *data)`
Appends an element to the end of the queue.
- void `priority_enqueue (queue_t *cue, pcb_t *data)`
Appends an element onto the tail of the given queue.
- `pcb_t * dequeue (queue_t *queue)`
Takes the PCB off of the head of the queue and moves head.
- `queue_t * construct_queue ()`
Creates a queue.
- void `destruct_queue (queue_t *queue)`
Destructs a queue and its contents.

4.5.1 Detailed Description

File to hold all queue functions.

4.5.2 Function Documentation

4.5.2.1 `queue_t* construct_queue ()`

Creates a queue.

Creates and allocates a queue and sets all variables correctly for initialization.

Returns

A pointer to a newly constructed queue.

4.5.2.2 `pcb_t* dequeue (queue_t * queue)`

Takes the PCB off of the head of the queue and moves head.

Takes care of freeing the node returns the head PCB

Parameters

<code>queue</code>	A pointer to a queue that you want to dequeue the first element from.
--------------------	---

Returns

A pointer to the dequeued PCB

4.5.2.3 `void destruct_queue (queue_t * queue)`

Destructs a queue and its contents.

De-allocates a queue and all of the elements within it. This function exists to avoid memory leaks.

Parameters

<i>queue</i>	A pointer to the queue you wish to deallocate.
--------------	--

4.5.2.4 void enqueue (queue_t * cue, pcb_t * data)

Appends an element to the end of the queue.

This function searches for the end of the queue and, adds the specified pcb to the end of the list.

Parameters

<i>que</i>	A pointer to a queue that the PCB will be inserted into.
<i>data</i>	A pointer to the PCB to insert into the queue.

4.5.2.5 void priority_enqueue (queue_t * cue, pcb_t * data)

Appends an element onto the tail of the given queue.

This function inserts the given data (a PCB) into the queue according to priority.

Parameters

<i>que</i>	A pointer to the queue to insert the data into.
<i>data</i>	a pointer to the PCB that is to be inserted.

Note

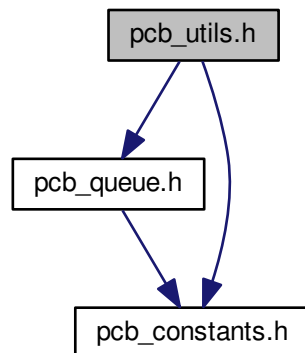
The data must point to a pcb with a valid priority.

4.6 pcb_utils.h File Reference

Utility functions for all PCBs.

```
#include "pcb_queue.h"
#include "pcb_constants.h"
```

Include dependency graph for pcb_utils.h:



Functions

- `pcb_t * allocate_pcb ()`
simply allocates space for a pcb and returns that pointer
- `pcb_t * setup_pcb (char *, PROCESS_CLASS, int priority)`
command to setup new PCB
- `int free_pcb (pcb_t *)`
frees the space for a pcb
- `pcb_t * find_pcb (char *pname)`
Finds a PCB in all queues.
- `int insert_pcb (pcb_t *)`
- `pcb_t * remove_pcb (char *pname)`
Removes a PCB by process name.
- `void init_queue ()`
- `queue_t * get_ready_queue ()`
Getter function for the ready queue.
- `queue_t * get_blocked_queue ()`
Getter function for the blocked queue.
- `queue_t * get_suspended_ready_queue ()`
Getter function for the suspended ready queue.
- `queue_t * get_suspended_blocked_queue ()`
Getter function for the suspended blocked queue.
- `void print_pcb_info (const pcb_t *pcb)`
Prints the passed pcb's info in a stylized manner.
- `const char * get_process_class_string (PROCESS_CLASS process_class)`
Returns a string corresponding to the process class enum.
- `const char * get_process_state_string (PROCESS_STATE process_state)`
Returns a string corresponding to the process state enum.

4.6.1 Detailed Description

Utility functions for all PCBs.

4.6.2 Function Documentation

4.6.2.1 `pcb_t* allocate_pcb ()`

simply allocates space for a pcb and returns that pointer

Returns

pointer to allocated pcb

4.6.2.2 `pcb_t* find_pcb (char * name)`

Finds a PCB in all queues.

Searches through all the system PCB queues to find a PCB with the specified process name given by pname.

Parameters

<i>pname</i>	The name of the process you want to find the PCB of.
--------------	--

Returns

A pointer to the pcb with the specified name or 'NULL' for not found.

4.6.2.3 `int free_pcb (pcb_t * p)`

frees the space for a pcb

Returns

Success or failure

4.6.2.4 `queue_t* get_blocked_queue ()`

Getter function for the blocked queue.

Returns

A pointer to the blocked queue

4.6.2.5 `const char* get_process_class_string (PROCESS_CLASS process_class)`

Returns a string corresponding to the process class enum.

Parameters

<i>process_class</i>	An enumeration variant of the PROCESS_CLASS enum
----------------------	--

Returns

A char pointer that is the enumeration name

4.6.2.6 `const char* get_process_state_string (PROCESS_STATE process_state)`

Returns a string corresponding to the process state enum.

Parameters

<i>process_state</i>	An enumeration variant of the PROCESS_STATE enum
----------------------	--

Returns

A char pointer that is the enumeration name

4.6.2.7 `queue_t* get_ready_queue ()`

Getter function for the ready queue.

Returns

A pointer to the ready queue

4.6.2.8 `queue_t* get_suspended_blocked_queue ()`

Getter function for the suspended blocked queue.

Returns

A pointer to the suspended blocked queue

4.6.2.9 `queue_t* get_suspended_ready_queue ()`

Getter function for the suspended ready queue.

Returns

A pointer to the suspended ready queue

4.6.2.10 void print_pcb_info (const pcb_t * pcb)

Prints the passed pcb's info in a stylized manner.

Example output

Process Name: PROC1 Process Class: system State: ready Priority: 1 Suspended: true

4.6.2.11 pcb_t* remove_pcb (char * name)

Removes a PCB by process name.

Searches through all system queues to find the PCB with the given name.

Parameters

<i>pname</i>	Name of the process you want to remove.
--------------	---

Returns

Success condition (boolean).

4.6.2.12 pcb_t* setup_pcb (char * pname, PROCESS_CLASS pclass, int priority)

command to setup new PCB

Returns

Success if the PCB was created, failure for anything else

4.7 pcb_wrangler.h File Reference

Initiates the creation of all queues.

Functions

- void **init_process_queues** ()

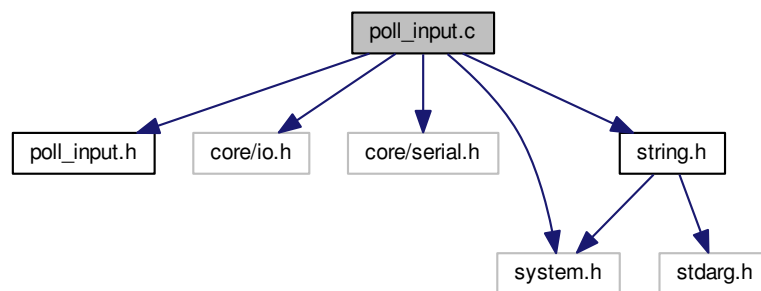
4.7.1 Detailed Description

Initiates the creation of all queues.

4.8 poll_input.c File Reference

The polling input file that allows user input.

```
#include "poll_input.h"
#include <core/io.h>
#include <core/serial.h>
#include <string.h>
#include <system.h>
Include dependency graph for poll_input.c:
```



Macros

- `#define BUFFER_LEN 100`

Functions

- `int input_available ()`
Checks for input on COM1.
- `int wait_for_input (int timeout)`
Loops N times to check for input.
- `int get_key ()`
Receives a key press, whether a full control sequence or simple character.
- `void move_cursor (int n)`
Moves the cursor n characters.
- `void print_after_cursor (const char *str)`
Prints text after the cursor without moving the cursor.
- `void delete_after_cursor ()`
Deletes all text after the cursor.
- `void memcpy (char *destination, const char *source, int n)`
Copies n bytes from one buffer to another.
- `int poll_input (char *buffer, int *length)`
Polls COM1 for input and puts it into buffer.

Variables

- const `ControlSequence control_sequences []`
A collection of known control sequences and what they mean.
- const int `TOLERANCE = 300`
Maximum amount of NOP cycles that can occur between two inputs from the same control sequence.
- const char `ESC = '\x1B'`
The escape character.
- const int `ALT_FLAG = 1 << 8`
The bit indicating a key from `get_key` was held with the ALT key.

4.8.1 Detailed Description

The polling input file that allows user input.

4.8.2 Function Documentation

4.8.2.1 `int get_key ()`

Receives a key press, whether a full control sequence or simple character.

Calls `inb(COM1)` to receive bytes. If a control sequence is detected then it is parsed according to the `control_sequences` array. If it was just a simple character like the A key. Then the char is sent as an int. Arrow keys and other control sequences are special numbers higher than 255 to differentiate themselves from the regular characters. The KEYS enum shows the special characters

Returns

Returns an int corresponding to the key

4.8.2.2 `int input_available ()`

Checks for input on COM1.

Returns

1 if input is available, 0 if it isn't.

4.8.2.3 `void memcpy (char * destination, const char * source, int n)`

Copies n bytes from one buffer to another.

Parameters

<i>destination</i>	Where to copy the bytes to.
<i>source</i>	Where to copy the bytes from.
<i>n</i>	How many bytes to copy.

4.8.2.4 void move_cursor (int *n*)

Moves the cursor *n* characters.

Parameters

<i>n</i>	How many characters to move the character, can be negative.
----------	---

4.8.2.5 int poll_input (char * *buffer*, int * *length*)

Polls COM1 for input and puts it into *buffer*.

An internal history is kept so the user can go through past commands

Parameters

<i>buffer</i>	a pointer to the buffer to put the user input into
<i>length</i>	a pointer to the length of buffer, will be modified to length of input

Returns

function status

4.8.2.6 void print_after_cursor (const char * *str*)

Prints text after the cursor without moving the cursor.

Parameters

<i>str</i>	A pointer to the string to print out
------------	--------------------------------------

4.8.2.7 int wait_for_input (int *timeout*)

Loops *N* times to check for input.

Calls NOP in a while loop at most *timeout* times until it returns.

Parameters

<i>timeout</i>	How many times to loop before we give up
----------------	--

Returns

how many times were left in the timeout

4.8.3 Variable Documentation

4.8.3.1 `const ControlSequence control_sequences[]`

Initial value:

```
= {
    {"A", UP_ARROW},
    {"B", DOWN_ARROW},
    {"C", RIGHT_ARROW},
    {"D", LEFT_ARROW},

    {"1~", HOME},
    {"2~", INSERT},
    {"3~", DELETE},
    {"4~", END},
    {"5~", PAGE_DOWN},
    {"6~", PAGE_UP},

    {"[A", F1},
    {"[B", F2},
    {"[C", F3},
    {"[D", F4},
    {"[E", F5},
    {"[17~", F6},
    {"[18~", F7},
    {"[19~", F8},
    {"[20~", F9},
    {"[21~", F10},
    {"[23~", F11},
    {"[24~", F12},

    {"", 0}
}
```

A collection of known control sequences and what they mean.

Control sequences are used to encode special input keys from the keyboard that aren't just a one byte character. They start with ESCAPE [and then a series of characters. This array holds the series of characters that comes after the bracket, along with the corresponding keyboard input. The keyboard inputs are from the KEYS enum.

4.8.3.2 `const int TOLERANCE = 300`

Maximum amount of NOP cycles that can occur between two inputs from the same control sequence.

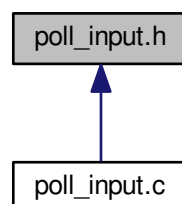
Note

This is entirely arbitrary and was just increased until things stopped being weird.

4.9 `poll_input.h` File Reference

The header file for the polling input.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [control_sequence](#)
A struct to hold key mappings.

Typedefs

- typedef struct [control_sequence](#) ControlSequence
A struct to hold key mappings.

Enumerations

- enum KEYS {
 BASE = 1024, UP_ARROW, DOWN_ARROW, RIGHT_ARROW,
 LEFT_ARROW, HOME, INSERT, DELETE,
 END, PAGE_UP, PAGE_DOWN, F1,
 F2, F3, F4, F5,
 F6, F7, F8, F9,
 F10, F11, F12 }

Functions

- int [poll_input](#) (char *buffer, int *length)
Polls COM1 for input and puts it into buffer.

4.9.1 Detailed Description

The header file for the polling input.

4.9.2 Typedef Documentation

4.9.2.1 typedef struct control_sequence ControlSequence

A struct to hold key mappings.

The [control_sequence](#) Struct is a custom struct that is designed to hold mappings between control sequence codes used to encode arrow keys. It also holds other special buttons.

Parameters

<i>code</i>	The special keyboard code name
<i>id</i>	The keyboard code value

4.9.3 Function Documentation

4.9.3.1 int poll_input (char * *buffer*, int * *length*)

Polls COM1 for input and puts it into buffer.

An internal history is kept so the user can go through past commands

Parameters

<i>buffer</i>	a pointer to the buffer to put the user input into
<i>length</i>	a pointer to the length of buffer, will be modified to length of input

Returns

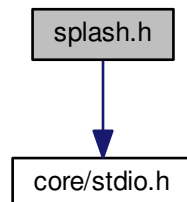
function status

4.10 splash.h File Reference

File to hold the splash screen.

```
#include <core/stdio.h>
```

Include dependency graph for splash.h:



Functions

- void **draw_splash** ()

4.10.1 Detailed Description

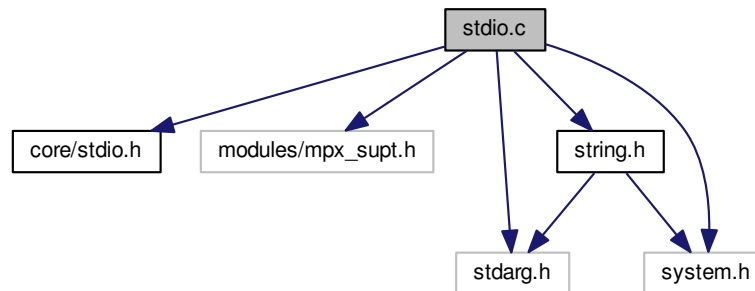
File to hold the splash screen.

4.11 stdio.c File Reference

Holds all implementation of standard I/O functions.

```
#include <core/stdio.h>
#include <modules/mpx_supt.h>
#include <stdarg.h>
#include <system.h>
#include <string.h>
```

Include dependency graph for stdio.c:



Functions

- int **printf** (char *form,...)
takes in a format string and prints it out to the DEFAULT_DEVICE
- int **puts** (char *buff)
prints out a string to DEFAULT_DEVICE

4.11.1 Detailed Description

Holds all implementation of standard I/O functions.

4.11.2 Function Documentation

4.11.2.1 int printf (char * form, ...)

takes in a format string and prints it out to the DEFAULT_DEVICE

c - character d/i - decimal integer x - hexadecimal integer s - string %% - percent sign Numbers can be included before the format specifier to declare alignment. i.e. %-10c = "c " A pad with 0s can also be added using a '0' directly after the percent i.e. %03c = "00c"

Parameters

<i>form</i>	character pointer to the format
<i>vaist</i>	variadic arguments to match the format (see brief)

Returns

0 for failure 1 for success

4.11.2.2 int puts (char * buff)

prints out a string to DEFAULT_DEVICE

Parameters

<i>buff</i>	string to print out
-------------	---------------------

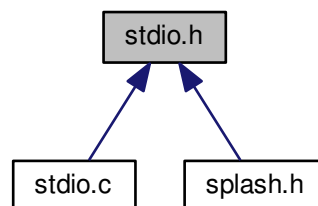
Returns

1

4.12 stdio.h File Reference

Holds all prototypes of standard I/O functions.

This graph shows which files directly or indirectly include this file:

**Functions**

- int [printf](#) (char *form,...)
takes in a format string and prints it out to the DEFAULT_DEVICE
- int [puts](#) (char *buffer)
prints out a string to DEFAULT_DEVICE

4.12.1 Detailed Description

Holds all prototypes of standard I/O functions.

4.12.2 Function Documentation

4.12.2.1 `int printf (char * form, ...)`

takes in a format string and prints it out to the DEFAULT_DEVICE

c - character d/i - decimal integer x - hexadecimal integer s - string %% - percent sign Numbers can be included before the format specifier to declare alignment. i.e. %-10c = "c " A pad with 0s can also be added using a '0' directly after the percent i.e. %03c = "00c"

Parameters

<i>form</i>	character pointer to the format
<i>valist</i>	variadic arguments to match the format (see brief)

Returns

0 for failure 1 for success

4.12.2.2 `int puts (char * buff)`

prints out a string to DEFAULT_DEVICE

Parameters

<i>buff</i>	string to print out
-------------	---------------------

Returns

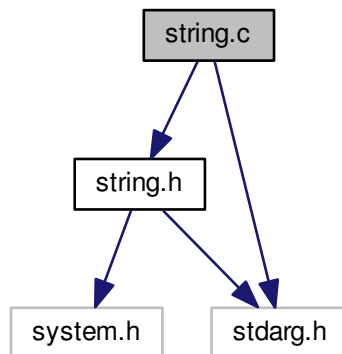
1

4.13 string.c File Reference

Holds all utility functions used to modify strings.

```
#include <string.h>
#include <stdarg.h>
```

Include dependency graph for string.c:



Macros

- `#define F_MINUS (1 << 0)`
- `#define F_PLUS (1 << 1)`
- `#define F_PERCENT (1 << 2)`
- `#define F_ZERO (1 << 3)`

Typedefs

- `typedef unsigned char BYTE`

Functions

- `int strlen (const char *s)`
- `char * strcpy (char *s1, const char *s2)`
- `int atoi (const char *s)`
- `int strcmp (const char *s1, const char *s2)`
- `char * strcat (char *s1, const char *s2)`
- `int isspace (const char *c)`
- `void * memset (void *s, int c, size_t n)`
- `char * strtok (char *s1, const char *s2)`
- `int isdigit (char c)`
Checks if char c is a digit.
- `char * reverse (char *str, int j)`
reverse a string from 0 to j
- `char * itoa (int num, char *str, int base)`
Converts integer to string.
- `char * sprintf_pad_helper (char *buffer, char pad, int fNum, int n, BYTE doAction)`
adds spaces where needed for the sprintf function
- `int sprintf_internal (char *buffer, char *format, va_list valist)`

Main implementation of the sprintf function.

- int `sprintf` (char *buffer, char *format,...)

Visible representation of the sprintf function.

- char `tolower` (char c)

Returns the lowercase representation of a character.

- char `toupper` (char c)

Returns the uppercase representation of a character.

- char * `trim` (char *str)

Returns a string with the beginning and ending whitespaces removed.

4.13.1 Detailed Description

Holds all utility functions used to modify strings.

4.13.2 Function Documentation

4.13.2.1 int isdigit (char c)

Checks if char c is a digit.

Parameters

<i>c</i>	character to check
----------	--------------------

Returns

is digit: 1; is not digit: 0;

4.13.2.2 char* itoa (int num, char * str, int base)

Converts integer to string.

Parameters

<i>num</i>	number to convert
<i>str</i>	string to store result in
<i>base</i>	base to convert to

Returns

pointer to str

4.13.2.3 char* reverse (char * str, int j)

reverse a string from 0 to j

Parameters

<i>str</i>	string to reverse
<i>j</i>	index to reverse str to

Returns

pointer to str

4.13.2.4 int sprintf (char * *buffer*, char * *format*, ...)

Visible representation of the sprintf function.

c - character d/i - decimal integer x - hexadecimal integer s - string %% - percent sign Numbers can be included before the format specifier to declare alignment. i.e. %-10c = "c " A pad with 0s can also be added using a '0' directly after the percent i.e. %03c = "00c"

Parameters

<i>buffer</i>	character pointer to store spaces to
<i>format</i>	format string with format specifiers
<i>valist</i>	variadic list with parameters matching the format

Returns

pointer to buffer

4.13.2.5 int sprintf_internal (char * *buffer*, char * *format*, va_list *valist*)

Main implementation of the sprintf function.

c - character d/i - decimal integer x - hexadecimal integer s - string %% - percent sign Numbers can be included before the format specifier to declare alignment. i.e. %-10c = "c " A pad with 0s can also be added using a '0' directly after the percent i.e. %03c = "00c"

Parameters

<i>buffer</i>	character pointer to store spaces to
<i>format</i>	format string with format specifiers
<i>valist</i>	variadic list with parameters matching the format

Returns

pointer to buffer

4.13.2.6 `char* sprintf_pad_helper (char * buffer, char pad, int fNum, int n, BYTE doAction)`

adds spaces where needed for the sprintf function

Parameters

<i>buffer</i>	character pointer to store spaces to
<i>pad</i>	what character to pad with
<i>fNum</i>	format number from sprintf
<i>n</i>	length of string that has been/will be added
<i>doAction</i>	boolean on whether or not to add the spaces

Returns

pointer to buffer

4.13.2.7 char tolower (char c)

Returns the lowercase representation of a character.

Parameters

<i>c</i>	character to return the lowercase representation of
----------	---

Returns

lowercase representation of c in ASCII

4.13.2.8 char toupper (char c)

Returns the uppercase representation of a character.

Parameters

<i>c</i>	character to return the uppercase representation of
----------	---

Returns

uppercase representation of c in ASCII

4.13.2.9 char* trim (char * str)

Returns a string with the beginning and ending whitespaces removed.

Parameters

<i>str</i>	the string to have white spaces removed from
------------	--

Returns

a sting with the begining and ending whitespaces removed

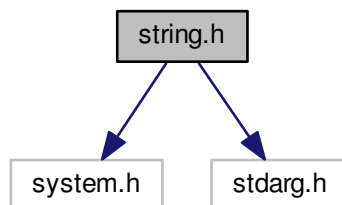
4.14 string.h File Reference

Holds all utility prototypes used to modify strings.

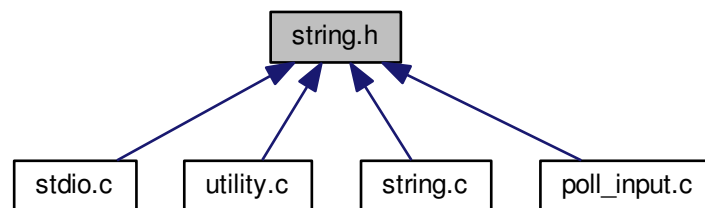
```
#include <system.h>
```

```
#include <stdarg.h>
```

Include dependency graph for string.h:



This graph shows which files directly or indirectly include this file:



Functions

- int **isspace** (const char *c)
- void * **memset** (void *s, int c, size_t n)
- char * **strcpy** (char *s1, const char *s2)
- char * **strcat** (char *s1, const char *s2)
- int **strlen** (const char *s)
- int **strcmp** (const char *s1, const char *s2)
- char * **strtok** (char *s1, const char *s2)
- int **isdigit** (char c)

- Checks if char c is a digit.*
- char * [reverse](#) (char *str, int j)
reverse a string from 0 to j
- int **atoi** (const char *s)
- char * [itoa](#) (int num, char *str, int base)
Converts integer to string.
- int [sprintf](#) (char *buffer, char *format,...)
Visible representation of the sprintf function.
- int [sprintf_internal](#) (char *buffer, char *format, va_list valist)
Main implementation of the sprintf function.
- char [tolower](#) (char c)
Returns the lowercase representation of a character.
- char [toupper](#) (char c)
Returns the uppercase representation of a character.
- char * [trim](#) (char *str)
Returns a string with the beginning and ending whitespaces removed.

4.14.1 Detailed Description

Holds all utility prototypes used to modify strings.

4.14.2 Function Documentation

4.14.2.1 int isdigit (char c)

Checks if char c is a digit.

Parameters

<i>c</i>	character to check
----------	--------------------

Returns

is digit: 1; is not digit: 0;

4.14.2.2 char* itoa (int num, char * str, int base)

Converts integer to string.

Parameters

<i>num</i>	number to convert
<i>str</i>	string to store result in
<i>base</i>	base to convert to

Returns

pointer to *str*

4.14.2.3 `char* reverse (char * str, int j)`

reverse a string from 0 to *j*

Parameters

<i>str</i>	string to reverse
<i>j</i>	index to reverse <i>str</i> to

Returns

pointer to *str*

4.14.2.4 `int sprintf (char * buffer, char * format, ...)`

Visible representation of the `sprintf` function.

c - character *d/i* - decimal integer *x* - hexadecimal integer *s* - string *%%* - percent sign Numbers can be included before the format specifier to declare alignment. i.e. `%-10c = "c "` A pad with 0s can also be added using a '0' directly after the percent i.e. `%03c = "00c"`

Parameters

<i>buffer</i>	character pointer to store spaces to
<i>format</i>	format string with format specifiers
<i>valist</i>	variadic list with parameters matching the format

Returns

pointer to *buffer*

4.14.2.5 `int sprintf_internal (char * buffer, char * format, va_list valist)`

Main implementation of the `sprintf` function.

c - character *d/i* - decimal integer *x* - hexadecimal integer *s* - string *%%* - percent sign Numbers can be included before the format specifier to declare alignment. i.e. `%-10c = "c "` A pad with 0s can also be added using a '0' directly after the percent i.e. `%03c = "00c"`

Parameters

<i>buffer</i>	character pointer to store spaces to
<i>format</i>	format string with format specifiers
<i>valist</i>	variadic list with parameters matching the format

Returns

pointer to buffer

4.14.2.6 char tolower (char *c*)

Returns the lowercase representation of a character.

Parameters

<i>c</i>	character to return the lowercase representation of
----------	---

Returns

lowercase representation of *c* in ASCII

4.14.2.7 char toupper (char *c*)

Returns the uppercase representation of a character.

Parameters

<i>c</i>	character to return the uppercase representation of
----------	---

Returns

uppercase representation of *c* in ASCII

4.14.2.8 char* trim (char * *str*)

Returns a string with the beginning and ending whitespaces removed.

Parameters

<i>str</i>	the string to have white spaces removed from
------------	--

Returns

a sting with the beginning and ending whitespaces removed

4.15 time.h File Reference

The header file for the date and time functions.

Data Structures

- struct [time](#)
A struct to all the time and date elements.

Macros

- #define **SECOND_REG** 0x00
- #define **MINUTE_REG** 0x02
- #define **HOURL_REG** 0x04
- #define **DAY_OF_MONTH_REG** 0x07
- #define **MONTH_REG** 0x08
- #define **CENTURY_REG** 0x32
- #define **YEAR_REG** 0x09
- #define **INDEX_REG** 0x70
- #define **DATA_REG** 0x71

Typedefs

- typedef struct [time](#) **time_h**

Enumerations

- enum **MONTH** {
 JANUARY = 1, **FEBRUARY**, **MARCH**, **APRIL**,
 MAY, **JUNE**, **JULY**, **AGUST**,
 SEPTEMBER, **OCTOBER**, **NOVEMBER**, **DECEMBER** }

Functions

- void [format_time](#) (char *dest, [time_h](#) *t)
Generates a string with a standard format of time.
- [time_h](#) [get_current_time](#) ()
Retrieves the current time in the Real Time Clock(RTC).
- int [set_current_time](#) ([time_h](#) time)
Sets the current time in the RTC.
- int [bcd_to_decimal](#) (int bcd)
Converts BCD values into decimal.

4.15.1 Detailed Description

The header file for the date and time functions.

4.15.2 Function Documentation

4.15.2.1 int [bcd_to_decimal](#) (int *bcd*)

Converts BCD values into decimal.

This function converts BCD values, to be a more code friendly decimal value.

Parameters

<i>bcd</i>	Value that is in BCD that needs to be a normal decimal value.
------------	---

Returns

The value of the BCD as an integer.

4.15.2.2 void format_time (char * *dest*, time_h * *t*)

Generates a string with a standard format of time.

Generates a string that contains all the data contained in a time_h. This form shows all data from largest timescale to smallest timescale.

Parameters

<i>dest</i>	Pointer to a string that is large enough to contain the output string
<i>time</i>	Pointer to the time to write into the destination string.

Returns

Return is through the 'dest' pointer.

Note

This is merely a convenience, as it is only an sprintf call.

4.15.2.3 time_h get_current_time ()

Retrieves the current time in the Real Time Clock(RTC).

Acquires data from the RTC, packaging the data into a time_h struct for ease of use.

Returns

Returns the current time represented as 6 values in a time_h struct.

4.15.2.4 int set_current_time (time_h *time*)

Sets the current time in the RTC.

Uses a time_h struct to set the data members of the RTC. This function also does error checking on valid times, including leap-years, valid days of months, etc., to ensure the given time is valid.

Parameters

<i>time</i>	A time_h struct containing the new time, as defined by the user.
-------------	--

Returns

If the operation was successful in boolean format (1 = true, 0 = false).

Note

This function also ensures that the date will be set in the correct order within the RTC.

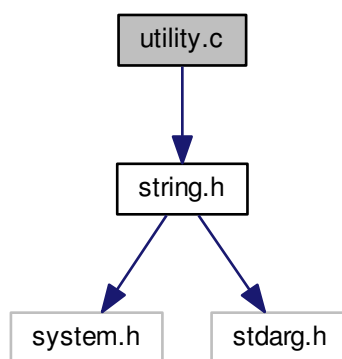
Setting a value in the input struct to a '-1' will skip the value in setting the time. Essentially, keeping the value as it was before. This is demonstrated in the commands.c file.

4.16 utility.c File Reference

Holds utility function implementations for this project.

```
#include <string.h>
```

Include dependency graph for utility.c:

**Functions**

- int [isnullorspace](#) (char test)
Determines if a passed character is a null or space.

4.16.1 Detailed Description

Holds utility function implementations for this project.

4.16.2 Function Documentation

4.16.2.1 int isnullorspace (char test)

Determines if a passed character is a null or space.

Parameters

<i>test</i>	character to test
-------------	-------------------

Returns

1 if space or null, 0 otherwise

4.17 utility.h File Reference

Holds utility function prototypes for this project.

Functions

- int [isnullorspace](#) (char test)
Determines if a passed character is a null or space.

4.17.1 Detailed Description

Holds utility function prototypes for this project.

4.17.2 Function Documentation

4.17.2.1 int isnullorspace (char *test*)

Determines if a passed character is a null or space.

Parameters

<i>test</i>	character to test
-------------	-------------------

Returns

1 if space or null, 0 otherwise

Index

A_FLAG
 commandUtils.h, 17
ALIAS, 5
allocate_pcb
 pcb_utils.h, 28
alphanum
 commandUtils.h, 17

B_FLAG
 commandUtils.h, 17
bcd_to_decimal
 time.h, 49

C_FLAG
 commandUtils.h, 17
CMD_SIZE
 commandUtils.h, 17
COMMAND, 5
cmd_blockPCB
 commands.h, 12
cmd_clear
 commands.h, 12
cmd_create_pcb
 commands.h, 13
cmd_date
 commands.h, 13
cmd_delete_pcb
 commands.h, 13
cmd_help
 commands.h, 13
cmd_resume
 commands.h, 14
cmd_set_priority_pcb
 commands.h, 14
cmd_shutdown
 commands.h, 14
cmd_suspend
 commands.h, 14
cmd_time
 commands.h, 14
cmd_unblock_pcb
 commands.h, 15
cmd_version
 commands.h, 15
command_handler.h, 11
commandUtils.h, 16
 A_FLAG, 17
 alphanum, 17
 B_FLAG, 17
 C_FLAG, 17
 CMD_SIZE, 17
 D_FLAG, 18
 E_FLAG, 18
 F_FLAG, 18
 G_FLAG, 18
 get_pvalue, 20
 gparamstr, 21
 H_FLAG, 18
 I_FLAG, 18
 J_FLAG, 18
 K_FLAG, 18
 L_FLAG, 18
 M_FLAG, 18
 N_FLAG, 19
 NO_FLAG, 19
 O_FLAG, 19
 P_FLAG, 19
 Q_FLAG, 19
 R_FLAG, 19
 S_FLAG, 19
 set_flags, 20
 set_flags_search_alias, 21
 T_FLAG, 19
 U_FLAG, 19
 V_FLAG, 19
 W_FLAG, 20
 X_FLAG, 20
 Y_FLAG, 20
 Z_FLAG, 20
commands.h, 11
 cmd_blockPCB, 12
 cmd_clear, 12
 cmd_create_pcb, 13
 cmd_date, 13
 cmd_delete_pcb, 13
 cmd_help, 13
 cmd_resume, 14
 cmd_set_priority_pcb, 14
 cmd_shutdown, 14
 cmd_suspend, 14
 cmd_time, 14
 cmd_unblock_pcb, 15
 cmd_version, 15
construct_queue
 pcb_queue.h, 24
control_sequence, 6
control_sequences
 poll_input.c, 34
ControlSequence

- poll_input.h, 35
- D_FLAG
 - commandUtils.h, 18
- dequeue
 - pcb_queue.h, 24
- destruct_queue
 - pcb_queue.h, 24
- E_FLAG
 - commandUtils.h, 18
- enqueue
 - pcb_queue.h, 26
- F_FLAG
 - commandUtils.h, 18
- find_pcb
 - pcb_utils.h, 28
- format_time
 - time.h, 50
- free_pcb
 - pcb_utils.h, 28
- G_FLAG
 - commandUtils.h, 18
- get_blocked_queue
 - pcb_utils.h, 28
- get_current_time
 - time.h, 50
- get_key
 - poll_input.c, 32
- get_process_class_string
 - pcb_utils.h, 28
- get_process_state_string
 - pcb_utils.h, 29
- get_pvalue
 - commandUtils.h, 20
- get_ready_queue
 - pcb_utils.h, 29
- get_suspended_blocked_queue
 - pcb_utils.h, 29
- get_suspended_ready_queue
 - pcb_utils.h, 29
- gparamstr
 - commandUtils.h, 21
- H_FLAG
 - commandUtils.h, 18
- HELP_PAGES, 7
- I_FLAG
 - commandUtils.h, 18
- input_available
 - poll_input.c, 32
- isdigit
 - string.c, 41
 - string.h, 46
- isnullorspace
 - utility.c, 51
 - utility.h, 52
- itoa
 - string.c, 41
 - string.h, 46
- J_FLAG
 - commandUtils.h, 18
- K_FLAG
 - commandUtils.h, 18
- L_FLAG
 - commandUtils.h, 18
- M_FLAG
 - commandUtils.h, 18
- memcpy
 - poll_input.c, 32
- move_cursor
 - poll_input.c, 33
- N_FLAG
 - commandUtils.h, 19
- NO_FLAG
 - commandUtils.h, 19
- node, 7
- node_t
 - pcb_constants.h, 23
- O_FLAG
 - commandUtils.h, 19
- P_FLAG
 - commandUtils.h, 19
- pcb, 8
- pcb_constants.h, 21
 - node_t, 23
- pcb_queue.h, 23
 - construct_queue, 24
 - dequeue, 24
 - destruct_queue, 24
 - enqueue, 26
 - priority_enqueue, 26
- pcb_utils.h, 26
 - allocate_pcb, 28
 - find_pcb, 28
 - free_pcb, 28
 - get_blocked_queue, 28
 - get_process_class_string, 28
 - get_process_state_string, 29
 - get_ready_queue, 29
 - get_suspended_blocked_queue, 29
 - get_suspended_ready_queue, 29
 - print_pcb_info, 29
 - remove_pcb, 30
 - setup_pcb, 30
- pcb_wrangler.h, 30
- poll_input
 - poll_input.c, 33
 - poll_input.h, 35
- poll_input.c, 31

- control_sequences, 34
- get_key, 32
- input_available, 32
- memcpy, 32
- move_cursor, 33
- poll_input, 33
- print_after_cursor, 33
- TOLERANCE, 34
- wait_for_input, 33
- poll_input.h, 34
 - ControlSequence, 35
 - poll_input, 35
- print_after_cursor
 - poll_input.c, 33
- print_pcb_info
 - pcb_utils.h, 29
- printf
 - stdio.c, 37
 - stdio.h, 39
- priority_enqueue
 - pcb_queue.h, 26
- puts
 - stdio.c, 38
 - stdio.h, 39
- Q_FLAG
 - commandUtils.h, 19
- queue, 9
- R_FLAG
 - commandUtils.h, 19
- remove_pcb
 - pcb_utils.h, 30
- reverse
 - string.c, 41
 - string.h, 47
- S_FLAG
 - commandUtils.h, 19
- set_current_time
 - time.h, 50
- set_flags
 - commandUtils.h, 20
- set_flags_search_alias
 - commandUtils.h, 21
- setup_pcb
 - pcb_utils.h, 30
- splash.h, 36
- sprintf
 - string.c, 42
 - string.h, 47
- sprintf_internal
 - string.c, 42
 - string.h, 47
- sprintf_pad_helper
 - string.c, 42
- stdio.c, 37
 - printf, 37
 - puts, 38
- stdio.h, 38
 - printf, 39
 - puts, 39
- string.c, 39
 - isdigit, 41
 - itoa, 41
 - reverse, 41
 - sprintf, 42
 - sprintf_internal, 42
 - sprintf_pad_helper, 42
 - tolower, 44
 - toupper, 44
 - trim, 44
- string.h, 45
 - isdigit, 46
 - itoa, 46
 - reverse, 47
 - sprintf, 47
 - sprintf_internal, 47
 - tolower, 48
 - toupper, 48
 - trim, 48
- T_FLAG
 - commandUtils.h, 19
- TOLERANCE
 - poll_input.c, 34
- time, 9
- time.h, 48
 - bcd_to_decimal, 49
 - format_time, 50
 - get_current_time, 50
 - set_current_time, 50
- tolower
 - string.c, 44
 - string.h, 48
- toupper
 - string.c, 44
 - string.h, 48
- trim
 - string.c, 44
 - string.h, 48
- U_FLAG
 - commandUtils.h, 19
- utility.c, 51
 - isnullspace, 51
- utility.h, 52
 - isnullspace, 52
- V_FLAG
 - commandUtils.h, 19
- W_FLAG
 - commandUtils.h, 20
- wait_for_input
 - poll_input.c, 33
- X_FLAG

commandUtils.h, [20](#)

Y_FLAG

commandUtils.h, [20](#)

Z_FLAG

commandUtils.h, [20](#)