

# Potato Operating System

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Data Structure Documentation</b>	<b>5</b>
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	time Struct Reference . . . . .	6
3.4.1	Detailed Description . . . . .	7
<b>4</b>	<b>File Documentation</b>	<b>9</b>
4.1	mpx_core/modules/m1/command_handler.c File Reference . . . . .	9
4.1.1	Detailed Description . . . . .	10
4.1.2	Macro Definition Documentation . . . . .	10
4.1.2.1	CMDSIZE . . . . .	10
4.1.3	Function Documentation . . . . .	10
4.1.3.1	search_commands(char *cmd) . . . . .	10
4.1.4	Variable Documentation . . . . .	10
4.1.4.1	commands . . . . .	10

4.2	<a href="#">mpx_core/modules/m1/command_handler.h File Reference</a>	10
4.2.1	<a href="#">Detailed Description</a>	11
4.3	<a href="#">mpx_core/modules/m1/commands.c File Reference</a>	11
4.3.1	<a href="#">Detailed Description</a>	13
4.3.2	<a href="#">Macro Definition Documentation</a>	14
4.3.2.1	<a href="#">A_FLAG</a>	14
4.3.2.2	<a href="#">alphanum</a>	14
4.3.2.3	<a href="#">B_FLAG</a>	14
4.3.2.4	<a href="#">C_FLAG</a>	14
4.3.2.5	<a href="#">CMD_SIZE</a>	14
4.3.2.6	<a href="#">D_FLAG</a>	15
4.3.2.7	<a href="#">E_FLAG</a>	15
4.3.2.8	<a href="#">F_FLAG</a>	15
4.3.2.9	<a href="#">G_FLAG</a>	15
4.3.2.10	<a href="#">H_FLAG</a>	15
4.3.2.11	<a href="#">I_FLAG</a>	15
4.3.2.12	<a href="#">J_FLAG</a>	15
4.3.2.13	<a href="#">K_FLAG</a>	15
4.3.2.14	<a href="#">L_FLAG</a>	15
4.3.2.15	<a href="#">M_FLAG</a>	15
4.3.2.16	<a href="#">N_FLAG</a>	16
4.3.2.17	<a href="#">O_FLAG</a>	16
4.3.2.18	<a href="#">P_FLAG</a>	16
4.3.2.19	<a href="#">Q_FLAG</a>	16
4.3.2.20	<a href="#">R_FLAG</a>	16
4.3.2.21	<a href="#">S_FLAG</a>	16
4.3.2.22	<a href="#">T_FLAG</a>	16
4.3.2.23	<a href="#">U_FLAG</a>	16
4.3.2.24	<a href="#">V_FLAG</a>	16
4.3.2.25	<a href="#">W_FLAG</a>	16

4.3.2.26	X_FLAG	17
4.3.2.27	Y_FLAG	17
4.3.2.28	Z_FLAG	17
4.3.3	Function Documentation	17
4.3.3.1	cmd_clear(char *params)	17
4.3.3.2	cmd_date(char *params)	17
4.3.3.3	cmd_help(char *params)	18
4.3.3.4	cmd_time(char *params)	18
4.3.3.5	cmd_version(char *params)	18
4.3.3.6	get_pvalue(char c)	19
4.3.3.7	set_flags(char *paramstr, int *flag, int num_aliases,...)	19
4.3.3.8	set_flags_search_alias(char *alias, int num_aliases, ALIAS aliases[])	19
4.4	mpx_core/modules/m1/commands.h File Reference	20
4.4.1	Detailed Description	20
4.4.2	Function Documentation	21
4.4.2.1	cmd_clear(char *params)	21
4.4.2.2	cmd_date(char *params)	22
4.4.2.3	cmd_help(char *params)	22
4.4.2.4	cmd_time(char *params)	22
4.4.2.5	cmd_version(char *params)	24
4.5	mpx_core/modules/m1/help.h File Reference	24
4.5.1	Detailed Description	25
4.5.2	Macro Definition Documentation	26
4.5.2.1	HELP	26
4.5.2.2	HELP_DATE	26
4.5.2.3	HELP_DATE_FULL	26
4.5.2.4	HELP_HELP	26
4.5.2.5	HELP_HELP_FULL	27
4.5.2.6	HELP_SHUTDOWN	27
4.5.2.7	HELP_SHUTDOWN_FULL	27

4.5.2.8	HELP_TIME	27
4.5.2.9	HELP_TIME_FULL	28
4.5.2.10	HELP_VERSION	28
4.5.2.11	HELP_VERSION_FULL	28
4.5.2.12	VERSION	28
4.5.2.13	VERSION_FULL	29
4.6	mpx_core/modules/m1/poll_input.c File Reference	29
4.6.1	Detailed Description	30
4.6.2	Function Documentation	30
4.6.2.1	get_key()	30
4.6.2.2	input_available()	30
4.6.2.3	memcpy(char *destination, const char *source, int n)	30
4.6.2.4	move_cursor(int n)	31
4.6.2.5	poll_input(char *buffer, int *length)	31
4.6.2.6	print_after_cursor(const char *str)	31
4.6.2.7	wait_for_input(int timeout)	31
4.6.3	Variable Documentation	32
4.6.3.1	control_sequences	32
4.6.3.2	TOLERANCE	32
4.7	mpx_core/modules/m1/poll_input.h File Reference	33
4.7.1	Detailed Description	33
4.7.2	Typedef Documentation	34
4.7.2.1	ControlSequence	34
4.7.3	Function Documentation	34
4.7.3.1	poll_input(char *buffer, int *length)	34
4.8	mpx_core/modules/m1/time.c File Reference	34
4.8.1	Detailed Description	35
4.8.2	Macro Definition Documentation	35
4.8.2.1	decimal_to_bcd	35
4.8.2.2	neg_safe_set	36
4.8.2.3	pull_data	36
4.8.3	Function Documentation	37
4.8.3.1	bcd_to_decimal(int bcd)	37
4.8.3.2	format_time(char *dest, time_h *time)	37
4.8.3.3	get_current_time()	38
4.8.3.4	set_current_time(time_h time)	38
4.9	mpx_core/modules/m1/time.h File Reference	38
4.9.1	Detailed Description	40
4.9.2	Function Documentation	40
4.9.2.1	bcd_to_decimal(int bcd)	40
4.9.2.2	format_time(char *dest, time_h *t)	40
4.9.2.3	get_current_time()	41
4.9.2.4	set_current_time(time_h time)	41







# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ALIAS</a>	A struct to hold command aliases . . . . .	5
<a href="#">COMMAND</a>	A struct to hold commands . . . . .	5
<a href="#">control_sequence</a>	A struct to hold key mappings . . . . .	6
<a href="#">time</a>	A struct to all the time and date elements . . . . .	6



## Chapter 2

# File Index

### 2.1 File List

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

mpx_core/modules/m1/ <a href="#">command_handler.c</a>	
The primary command handler for the Operating System . . . . .	9
mpx_core/modules/m1/ <a href="#">command_handler.h</a>	
The header file for the command handler for the Operating System . . . . .	10
mpx_core/modules/m1/ <a href="#">commands.c</a>	
This file contains all the commands that will be used by the command handler . . . . .	11
mpx_core/modules/m1/ <a href="#">commands.h</a>	
The header file for <a href="#">commands.c</a> . . . . .	20
mpx_core/modules/m1/ <a href="#">help.h</a>	
The header file that contains all the macros for the help and version commands . . . . .	24
mpx_core/modules/m1/ <a href="#">poll_input.c</a>	
The polling input file that allows user input . . . . .	29
mpx_core/modules/m1/ <a href="#">poll_input.h</a>	
The header file for the polling input . . . . .	33
mpx_core/modules/m1/ <a href="#">time.c</a>	
The file that contains all the date and time system functions . . . . .	34
mpx_core/modules/m1/ <a href="#">time.h</a>	
The header file for the date and time functions . . . . .	38



## Chapter 3

# Data Structure Documentation

### 3.1 ALIAS Struct Reference

A struct to hold command aliases.

#### 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:

- [mpx\\_core/modules/m1/commands.c](#)

### 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:

- [mpx\\_core/modules/m1/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:

- [mpx\\_core/modules/m1/poll\\_input.h](#)

## 3.4 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**

#### 3.4.1 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:

- `mpx_core/modules/m1/time.h`





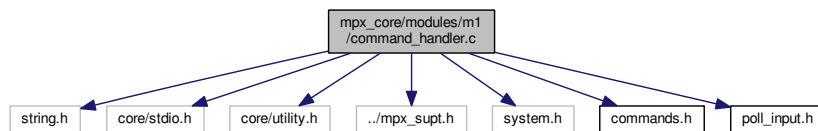
## Chapter 4

# File Documentation

### 4.1 mpx\_core/modules/m1/command\_handler.c File Reference

The primary command handler for the Operating System.

```
#include <string.h>
#include <core/stdio.h>
#include <core/utility.h>
#include "../mpx_supt.h"
#include "commands.h"
#include "poll_input.h"
Include dependency graph for command_handler.c:
```



### Data Structures

- struct `COMMAND`  
*A struct to hold commands.*

### Macros

- `#define CMDSIZE 100`  
*The command input buffer.*

### Functions

- int `search_commands` (char \*cmd)  
*Finds which command in the global COMMANDS array.*
- int `command_handler` ()  
*Entry point for the command handler.*

## Variables

- [COMMAND commands \[\]](#)

*Array of COMMANDS that are supported.*

### 4.1.1 Detailed Description

The primary command handler for the Operating System.

### 4.1.2 Macro Definition Documentation

#### 4.1.2.1 #define CMDSIZE 100

The command input buffer.

This is 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.

### 4.1.3 Function Documentation

#### 4.1.3.1 int search\_commands ( char \* cmd )

Finds which command in the global COMMANDS array.

#### Parameters

<i>cmd</i>	cmd typed by user
------------	-------------------

### 4.1.4 Variable Documentation

#### 4.1.4.1 COMMAND commands[]

#### Initial value:

```
= {
  {"help", &cmd_help},
  {"version", &cmd_version},
  {"date", &cmd_date},
  {"time", &cmd_time},
  {"clear", &cmd_clear},
  {NULL, NULL}
}
```

Array of COMMANDS that are supported.

## 4.2 mp\_x\_core/modules/m1/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.*

### 4.2.1 Detailed Description

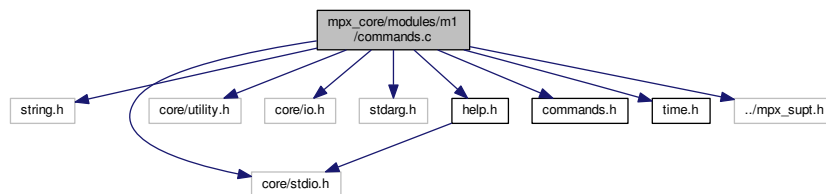
The header file for the command handler for the Operating System.

## 4.3 mpx\_core/modules/m1/commands.c File Reference

This file contains all the commands that will be used by the command handler.

```
#include <string.h>
#include <core/stdio.h>
#include <core/utility.h>
#include <core/io.h>
#include <stdarg.h>
#include "help.h"
#include "commands.h"
#include "time.h"
#include "../mpx_supt.h"
```

Include dependency graph for commands.c:



## Data Structures

- struct `ALIAS`  
*A struct to hold command aliases.*

## Macros

- #define `CMDSIZE` 100  
*The command input buffer.*
- #define `SUCCESS` 0  
*Macro to return a 0 on success.*
- #define `FAILURE` -1  
*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 [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.*
- 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\\_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\\_clear](#) (char \*params)  
*clears the screen and sets the pointer at home*

## Variables

- char `gparamstr` [CMD\_SIZE]  
*A string to hold the command input up to the max command size.*
- char \* `gparams` [26]  
*Will hold all the string pointers.*

### 4.3.1 Detailed Description

This file contains all the commands that will be used by the command handler.

sets flags based on param string, flags and num aliases

Gets value of specific flag.

Used as a helper function for set\_flags.

clears the screen and sets the pointer at home

#### Parameters

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

#### Returns

SUCCESS or FAILURE

#### 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

Usage: `get_pvalue('a');`

#### Parameters

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

#### Returns

value after the flag specified

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

**4.3.2 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.

A flag binary bit shift macro 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**

<i>c</i>	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 ammount 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 O\_FLAG (1 << 14)**

O flag binary bit shift macro

**4.3.2.18 #define P\_FLAG (1 << 15)**

P flag binary bit shift macro

**4.3.2.19 #define Q\_FLAG (1 << 16)**

Q flag binary bit shift macro

**4.3.2.20 #define R\_FLAG (1 << 17)**

R flag binary bit shift macro

**4.3.2.21 #define S\_FLAG (1 << 18)**

S flag binary bit shift macro

**4.3.2.22 #define T\_FLAG (1 << 19)**

T flag binary bit shift macro

**4.3.2.23 #define U\_FLAG (1 << 20)**

U flag binary bit shift macro

**4.3.2.24 #define V\_FLAG (1 << 21)**

V flag binary bit shift macro

**4.3.2.25 #define W\_FLAG (1 << 22)**

W flag binary bit shift macro



#### 4.3.2.26 #define X\_FLAG (1 << 24)

X flag binary bit shift macro

#### 4.3.2.27 #define Y\_FLAG (1 << 23)

Y flag binary bit shift macro

#### 4.3.2.28 #define Z\_FLAG (1 << 25)

Z flag binary bit shift macro

### 4.3.3 Function Documentation

#### 4.3.3.1 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

#### 4.3.3.2 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

#### 4.3.3.3 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

#### 4.3.3.4 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

#### 4.3.3.5 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 shortned 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

#### 4.3.3.6 `char * get_pvalue ( char c )`

Gets value of specific flag.

Usage: `get_pvalue('a');`

### Parameters

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

### Returns

value after the flag specified

#### 4.3.3.7 `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

#### 4.3.3.8 `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

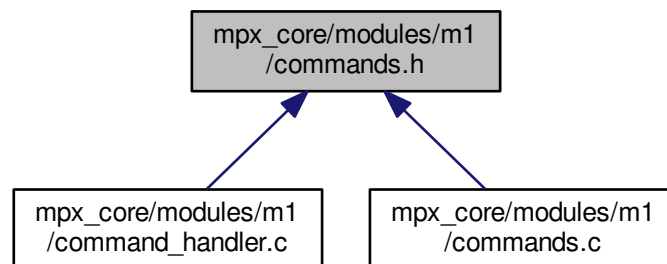
**Returns**

character of flag that it found

**4.4 mpx\_core/modules/m1/commands.h File Reference**

The header file for [commands.c](#).

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

**Macros**

- `#define SUCCESS 0`  
Macro to return a 0 on success.

**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)`
- `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

**4.4.1 Detailed Description**

The header file for [commands.c](#).

## 4.4.2 Function Documentation

### 4.4.2.1 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

**4.4.2.2 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

**4.4.2.3 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

**4.4.2.4 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

**4.4.2.5 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 shortned 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**

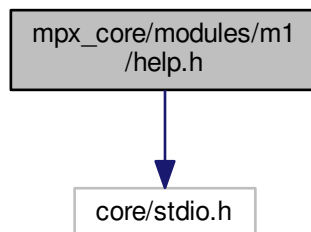
A version page

**4.5 mpx\_core/modules/m1/help.h File Reference**

The header file that contains all the macros for the help and version commands.

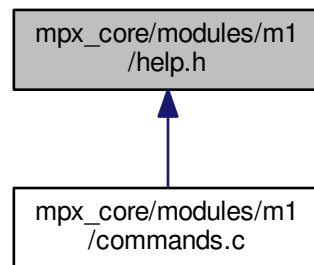
```
#include <core/stdio.h>
```

Include dependency graph for help.h:





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



## Macros

- `#define VERSION`  
Macro to print the short version.
- `#define VERSION_FULL`  
Macro to print the full version.
- `#define HELP`  
Macro to print the list of commands that have help pages.
- `#define HELP_HELP`  
Macro to print the shortned help page for comamnd help.
- `#define HELP_HELP_FULL`  
Macro to print the full help page for command help.
- `#define HELP_VERSION`  
Macro to print the shortned help page for command version.
- `#define HELP_VERSION_FULL`  
Macro to print the full help page for command version.
- `#define HELP_SHUTDOWN`  
Macro to print the shortned help page for command shutdown.
- `#define HELP_SHUTDOWN_FULL`  
Macro to print the full help page for command shutdown.
- `#define HELP_DATE`  
Macro to print the shortened help page for command date.
- `#define HELP_DATE_FULL`  
Macro to print the full help page for command date.
- `#define HELP_TIME`  
Macro to print the shortned help page for command time.
- `#define HELP_TIME_FULL`  
Macro to print the full help page for command time.

### 4.5.1 Detailed Description

The header file that contains all the macros for the help and version commands.

## 4.5.2 Macro Definition Documentation

### 4.5.2.1 #define HELP

#### Value:

```
puts(\
    "You can request a help page for the following commands"\
    " using help <cmd name>\n"\
    "\thelp\n"\
    "\tversion\n"\
    "\tshutdown\n"\
    "\tdate\n"\
    "\ttime"\
);
```

Macro to print the list of commands that have help pages.

### 4.5.2.2 #define HELP\_DATE

#### Value:

```
puts(\
    "Display date."\
    " Use flag [-f | --full] for more information"\
);
```

Macro to print the shortened help page for command date.

### 4.5.2.3 #define HELP\_DATE\_FULL

#### Value:

```
puts(\
    "Usage:\n"\
    "\tdate [-s | --set]\n"\
    "Flags:\n"\
    "\t[-s | --set] - Set the date in DD/MM/YYYY\n"\
    "\t\tWhere all values are integers\n"\
    "Example:\n"\
    "\tdate -s 08/24/1994\n"\
    "\tdate --set 01/01/2019"\
);
```

Macro to print the full help page for command date.

### 4.5.2.4 #define HELP\_HELP

#### Value:

```
puts(\
    "View help pages for individual commands."\
    " Use flag [-f | --full] for more information"\
);
```

Macro to print the shortned help page for comamnd help.

#### 4.5.2.5 #define HELP\_HELP\_FULL

**Value:**

```
puts(\
    "Usage:\n"\
    "\thelp [-c | --command] <command> [-f | --full]\n"\
    "\n"\
    "Flags:\n"\
    "\t[-c | --command] - Show the help page for a certain command\n"\
    "\t[-f | --full] - Show implementation and flags for each command\n"\
    "\t\twith explanations\
");
```

Macro to print the full help page for command help.

#### 4.5.2.6 #define HELP\_SHUTDOWN

**Value:**

```
puts(\
    "Shutdown the POS System.\n"\
    "Use flag [-f | --full] for more information\
");
```

Macro to print the shortned help page for command shutdown.

#### 4.5.2.7 #define HELP\_SHUTDOWN\_FULL

**Value:**

```
puts(\
    "Usage:\n"\
    "\tshutdown\n"\
    "Flags:\n"\
    "\tNone\n"\
    "Notes:\n"\
    "\tMust confirm with Yes before shutdown\
");
```

Macro to print the full help page for command shutdown.

#### 4.5.2.8 #define HELP\_TIME

**Value:**

```
puts(\
    "Display time.\n"\
    "Use flag [-f | --full] for more information\
");
```

Macro to print the shortned help page for command time.

#### 4.5.2.9 #define HELP\_TIME\_FULL

**Value:**

```
puts(\
    "Usage:\n"\
    "\tdate [-s | --set]\n"\
    "Flags:\n"\
    "\t[-s | --set] - Set the time in HH:MM:SS\n"\
    "\t\tWhere all values are integers and using 24 hour time\n"\
    "Example:\n"\
    "\tttime -s 12:24:32\n"\
    "\tttime --set 16:02:00"\
);
```

Macro to print the full help page for command time.

#### 4.5.2.10 #define HELP\_VERSION

**Value:**

```
puts(\
    "Display version information."\
    "Use flag [-f | --full] for more information"\
);
```

Macro to print the shortned help page for command version.

#### 4.5.2.11 #define HELP\_VERSION\_FULL

**Value:**

```
puts(\
    "Usage:\n"\
    "\tversion [-f | --full]\n"\
    "Flags:\n"\
    "\t[-f | --full] - Show entire version"\
);
```

Macro to print the full help page for command version.

#### 4.5.2.12 #define VERSION

**Value:**

```
puts(\
    "Version 1.0"\
);
```

Macro to print the short version.

## 4.5.2.13 #define VERSION\_FULL

## Value:

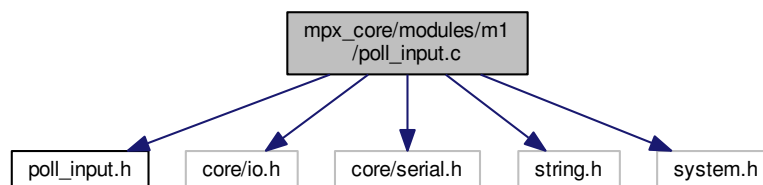
```
puts (\
    "Version 1.0\n"\
    "Module one\n"\
    "Developers:\n"\
    "\tHasan Ibraheem\n"\
    "\tHenry Vos\n"\
    "\tJay Kmetz\n"\
    "\tNicholas Fryer"\
);
```

Macro to print the full version.

## 4.6 mpx\_core/modules/m1/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.*

### 4.6.1 Detailed Description

The polling input file that allows user input.

### 4.6.2 Function Documentation

#### 4.6.2.1 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

#### 4.6.2.2 int `input_available` ( )

Checks for input on COM1.

#### Returns

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

#### 4.6.2.3 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.

4.6.2.4 void move\_cursor ( int *n* )

Moves the cursor *n* characters.

## Parameters

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

4.6.2.5 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

4.6.2.6 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
------------	--------------------------------------

4.6.2.7 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

**4.6.3 Variable Documentation****4.6.3.1 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.

**4.6.3.2 const int TOLERANCE = 300**

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

**Note**

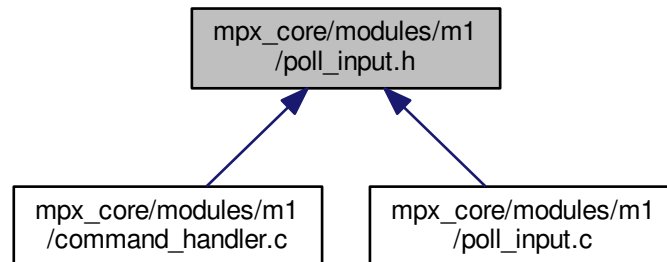
This is entirely arbitrary and was just increased until things stopped being weird.



## 4.7 mpx\_core/modules/m1/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.

#### 4.7.1 Detailed Description

The header file for the polling input.

## 4.7.2 Typedef Documentation

### 4.7.2.1 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

## 4.7.3 Function Documentation

### 4.7.3.1 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

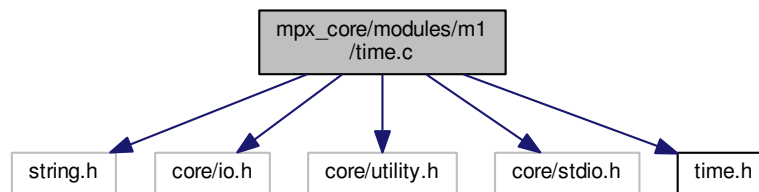
function status

## 4.8 mpx\_core/modules/m1/time.c File Reference

The file that contains all the date and time system functions.

```
#include <string.h>
#include <core/io.h>
#include <core/utility.h>
#include <core/stdio.h>
#include "time.h"
```

Include dependency graph for time.c:



## Macros

- `#define pull_data(val, loc)`  
Macro aquires data from a RTC register and converts the output from BCD to decimal.
- `#define decimal_to_bcd(val) ((val/10)<<4 | (val%10))`  
Simple macro to convert values into BCD format to write time to the RTC.
- `#define neg_safe_set(in, loc)`  
Wrtes a value to a RTC register.

## Functions

- `int bcd_to_decimal (int bcd)`  
Converts BCD values into decimal.
- `void format_time (char *dest, time_h *time)`  
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.

### 4.8.1 Detailed Description

The file that contains all the date and time system functions.

### 4.8.2 Macro Definition Documentation

#### 4.8.2.1 `#define decimal_to_bcd( val ) ((val/10)<<4 | (val%10))`

Simple macro to convert values into BCD format to write time to the RTC.

This is used to convert a value that could be defined as a literal or calculated by code, into BCD so that writing to the RTC is correct, and that it can keep time. This macro is nested in `neg_safe_set`.

**Parameters**

<i>val</i>	The value to be converted into BCD format.
------------	--

**Returns**

Returns the BCD of the given value.

**4.8.2.2 #define neg\_safe\_set( in, loc )****Value:**

```
{\
  if (in > -1)\
  {\
    outb(INDEX_REG, loc);\
    outb(DATA_REG, decimal_to_bcd(in));\
  }\
}
```

Writes a value to a RTC register.

Writes a decimal value to a RTC register. This converts the given value to BCD, and then writes it into the correct data location for the RTC to recognise what is being set. Uses outb to set the location of the data that is being written, and writing the actual data.

**Parameters**

<i>in</i>	The value that is being written to the RTC.
<i>loc</i>	Type of data being requested, this is using the macros specified in <a href="#">time.h</a> as YEAR_REG, MONTH_REG, etc.

**Returns**

void

**Warning**

Do not use values that are not specified as 'locations' as the loc field.  
'in' should not be BCD, it should be a normal value.

**4.8.2.3 #define pull\_data( val, loc )****Value:**

```
{\
  outb(INDEX_REG, loc);\
  char temp = inb(DATA_REG);\
  val = bcd_to_decimal(temp);\
  if (val < 0)\
  {\
    puts("\033[31mTIME ERROR: BCD conversion error in "#val".\033[0m");\
    return (time_h){-1,-1,-1,-1,-1,-1};\
  }\
}
```

Macro acquires data from a RTC register and converts the output from BCD to decimal.

This macro is used by the `get_time` function to acquire the RTC value and convert it into a usable decimal. The data at the location that is specified is written to the given vlaue. The function `outb` is used to select the type of information that is going to be written to the value, this is years, months, etc.. The bcd to base 2 conversion happens within the `bcd_to_decimal` macro.

#### Parameters

<i>val</i>	Value that the requested data is to be stored into. This data must not be a pointer.
<i>loc</i>	Type of data being requested, this is using the macros specified in <a href="#">time.h</a> as YEAR_REG, MONTH_REG, etc.

#### Returns

void

#### Warning

Do not use values that are not specifed as 'locations' as the `loc` field.

### 4.8.3 Function Documentation

#### 4.8.3.1 `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.

#### 4.8.3.2 `void format_time ( char * dest, time_h * time )`

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.

**4.8.3.3 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.

**4.8.3.4 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).

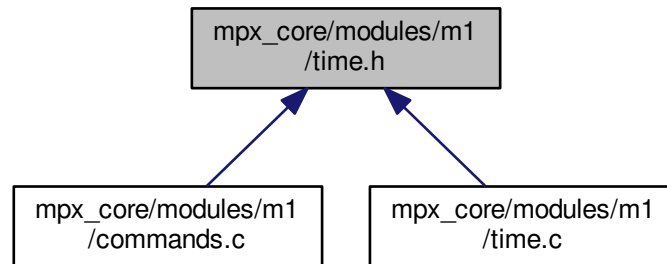
**Note**

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.

## 4.9 mpx\_core/modules/m1/time.h File Reference

The header file for the date and time functions.

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



## Data Structures

- struct [time](#)

*A struct to all the time and date elements.*

## Macros

- `#define SECOND_REG 0x00`
- `#define MINUTE_REG 0x02`
- `#define HOUR_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.*

### 4.9.1 Detailed Description

The header file for the date and time functions.

### 4.9.2 Function Documentation

#### 4.9.2.1 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.

#### 4.9.2.2 void `format_time` ( char \* *dest*, `time_h` \* *time* )

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.

**4.9.2.3 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.

**4.9.2.4 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.



# Index

A\_FLAG  
    commands.c, 14

ALIAS, 5

alphanum  
    commands.c, 14

B\_FLAG  
    commands.c, 14

bcd\_to\_decimal  
    time.c, 37  
    time.h, 40

C\_FLAG  
    commands.c, 14

CMDSIZE  
    command\_handler.c, 10  
    commands.c, 14

COMMAND, 5

cmd\_clear  
    commands.c, 17  
    commands.h, 21

cmd\_date  
    commands.c, 17  
    commands.h, 22

cmd\_help  
    commands.c, 17  
    commands.h, 22

cmd\_time  
    commands.c, 18  
    commands.h, 22

cmd\_version  
    commands.c, 18  
    commands.h, 24

command\_handler.c  
    CMDSIZE, 10  
    commands, 10  
    search\_commands, 10

commands  
    command\_handler.c, 10

commands.c  
    A\_FLAG, 14  
    alphanum, 14  
    B\_FLAG, 14  
    C\_FLAG, 14  
    CMDSIZE, 14  
    cmd\_clear, 17  
    cmd\_date, 17  
    cmd\_help, 17  
    cmd\_time, 18  
    cmd\_version, 18  
    D\_FLAG, 14  
    E\_FLAG, 15  
    F\_FLAG, 15  
    G\_FLAG, 15  
    get\_pvalue, 19  
    H\_FLAG, 15  
    I\_FLAG, 15  
    J\_FLAG, 15  
    K\_FLAG, 15  
    L\_FLAG, 15  
    M\_FLAG, 15  
    N\_FLAG, 15  
    O\_FLAG, 16  
    P\_FLAG, 16  
    Q\_FLAG, 16  
    R\_FLAG, 16  
    S\_FLAG, 16  
    set\_flags, 19  
    set\_flags\_search\_alias, 19  
    T\_FLAG, 16  
    U\_FLAG, 16  
    V\_FLAG, 16  
    W\_FLAG, 16  
    X\_FLAG, 16  
    Y\_FLAG, 17  
    Z\_FLAG, 17  
    commands.h  
        cmd\_clear, 21  
        cmd\_date, 22  
        cmd\_help, 22  
        cmd\_time, 22  
        cmd\_version, 24  
    control\_sequence, 6  
    control\_sequences  
        poll\_input.c, 32  
    ControlSequence  
        poll\_input.h, 34

D\_FLAG  
    commands.c, 14

decimal\_to\_bcd  
    time.c, 35

E\_FLAG  
    commands.c, 15

F\_FLAG  
    commands.c, 15

format\_time  
    time.c, 37

- time.h, 40
- G\_FLAG
  - commands.c, 15
- get\_current\_time
  - time.c, 38
  - time.h, 41
- get\_key
  - poll\_input.c, 30
- get\_pvalue
  - commands.c, 19
- H\_FLAG
  - commands.c, 15
- HELP\_DATE\_FULL
  - help.h, 26
- HELP\_DATE
  - help.h, 26
- HELP\_HELP\_FULL
  - help.h, 26
- HELP\_HELP
  - help.h, 26
- HELP\_SHUTDOWN\_FULL
  - help.h, 27
- HELP\_SHUTDOWN
  - help.h, 27
- HELP\_TIME\_FULL
  - help.h, 27
- HELP\_TIME
  - help.h, 27
- HELP\_VERSION\_FULL
  - help.h, 28
- HELP\_VERSION
  - help.h, 28
- HELP
  - help.h, 26
- help.h
  - HELP\_DATE\_FULL, 26
  - HELP\_DATE, 26
  - HELP\_HELP\_FULL, 26
  - HELP\_HELP, 26
  - HELP\_SHUTDOWN\_FULL, 27
  - HELP\_SHUTDOWN, 27
  - HELP\_TIME\_FULL, 27
  - HELP\_TIME, 27
  - HELP\_VERSION\_FULL, 28
  - HELP\_VERSION, 28
  - HELP, 26
  - VERSION\_FULL, 28
  - VERSION, 28
- I\_FLAG
  - commands.c, 15
- input\_available
  - poll\_input.c, 30
- J\_FLAG
  - commands.c, 15
- K\_FLAG
  - commands.c, 15
- L\_FLAG
  - commands.c, 15
- M\_FLAG
  - commands.c, 15
- memcpy
  - poll\_input.c, 30
- move\_cursor
  - poll\_input.c, 31
- mpx\_core/modules/m1/command\_handler.c, 9
- mpx\_core/modules/m1/command\_handler.h, 10
- mpx\_core/modules/m1/commands.c, 11
- mpx\_core/modules/m1/commands.h, 20
- mpx\_core/modules/m1/help.h, 24
- mpx\_core/modules/m1/poll\_input.c, 29
- mpx\_core/modules/m1/poll\_input.h, 33
- mpx\_core/modules/m1/time.c, 34
- mpx\_core/modules/m1/time.h, 38
- N\_FLAG
  - commands.c, 15
- neg\_safe\_set
  - time.c, 36
- O\_FLAG
  - commands.c, 16
- P\_FLAG
  - commands.c, 16
- poll\_input
  - poll\_input.c, 31
  - poll\_input.h, 34
- poll\_input.c
  - control\_sequences, 32
  - get\_key, 30
  - input\_available, 30
  - memcpy, 30
  - move\_cursor, 31
  - poll\_input, 31
  - print\_after\_cursor, 31
  - TOLERANCE, 32
  - wait\_for\_input, 31
- poll\_input.h
  - ControlSequence, 34
  - poll\_input, 34
- print\_after\_cursor
  - poll\_input.c, 31
- pull\_data
  - time.c, 36
- Q\_FLAG
  - commands.c, 16
- R\_FLAG
  - commands.c, 16
- S\_FLAG
  - commands.c, 16

- search\_commands
  - command\_handler.c, [10](#)
- set\_current\_time
  - time.c, [38](#)
  - time.h, [41](#)
- set\_flags
  - commands.c, [19](#)
- set\_flags\_search\_alias
  - commands.c, [19](#)
- T\_FLAG
  - commands.c, [16](#)
- TOLERANCE
  - poll\_input.c, [32](#)
- time, [6](#)
- time.c
  - bcd\_to\_decimal, [37](#)
  - decimal\_to\_bcd, [35](#)
  - format\_time, [37](#)
  - get\_current\_time, [38](#)
  - neg\_safe\_set, [36](#)
  - pull\_data, [36](#)
  - set\_current\_time, [38](#)
- time.h
  - bcd\_to\_decimal, [40](#)
  - format\_time, [40](#)
  - get\_current\_time, [41](#)
  - set\_current\_time, [41](#)
- U\_FLAG
  - commands.c, [16](#)
- V\_FLAG
  - commands.c, [16](#)
- VERSION\_FULL
  - help.h, [28](#)
- VERSION
  - help.h, [28](#)
- W\_FLAG
  - commands.c, [16](#)
- wait\_for\_input
  - poll\_input.c, [31](#)
- X\_FLAG
  - commands.c, [16](#)
- Y\_FLAG
  - commands.c, [17](#)
- Z\_FLAG
  - commands.c, [17](#)