Arduino Resource Based File System

Generated by Doxygen 1.8.9.1

Wed Aug 19 2015 01:07:47

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 rbfs_global_flags_t Struct Reference

`#include <rbfs.h>`

Public Attributes

- uint8_t driver_mouted

3.1.1 Detailed Description

Definition at line 142 of file rbfs.h.

3.1.2 Member Data Documentation

3.1.2.1 uint8_t rbfs_global_flags_t::driver_mouted

Definition at line 143 of file rbfs.h.

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

- rbfs.h

3.2 rbfs_resource_t Struct Reference

`#include <rbfs.h>`

Public Attributes

- rbfs_resource_descriptor_t resource_descriptor
- rbfs_cluster_t first_cluster
- rbfs_cluster_t current_cluster
- uint8_t cluster_offset
- uint16_t size
- uint16_t current_position
- uint8_t flags

3.2.1 Detailed Description

Definition at line 132 of file rbfs.h.

3.2.2 Member Data Documentation

3.2.2.1 uint8_t rbfs_resource_t::cluster_offset

Definition at line 136 of file rbfs.h.

3.2.2.2 rbfs_cluster_t rbfs_resource_t::current_cluster

Definition at line 135 of file rbfs.h.

3.2.2.3 uint16_t rbfs_resource_t::current_position

Definition at line 138 of file rbfs.h.

3.2.2.4 rbfs_cluster_t rbfs_resource_t::first_cluster

Definition at line 134 of file rbfs.h.

3.2.2.5 uint8_t rbfs_resource_t::flags

Definition at line 139 of file rbfs.h.

3.2.2.6 rbfs_resource_descriptor_t rbfs_resource_t::resource_descriptor

Definition at line 133 of file rbfs.h.

3.2.2.7 uint16_t rbfs_resource_t::size

Definition at line 137 of file rbfs.h.

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

- rbfs.h

3.3 rbfs_stat_t Struct Reference

`#include <rbfs.h>`

Public Attributes

- uint8_t flags

3.3.1 Detailed Description

Definition at line 107 of file rbfs.h.

3.3.2 Member Data Documentation

3.3.2.1 uint8_t rbfs_stat_t::flags

Definition at line 108 of file rbfs.h.

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

- rbfs.h

3.4 rbfs_t Struct Reference

`#include <rbfs.h>`

Public Attributes

- rbfs_driver_t driver
- uint16_t memory_size
- rbfs_memory_address_t resource_descriptor_table_address
- rbfs_memory_address_t cluster_table_address
- uint16_t sizeof_resource_descriptor_table
- uint16_t sizeof_cluster_table
- uint8_t sizeof_resource_descriptor
- uint8_t sizeof_cluster
- uint8_t resource_descriptor_count
- uint8_t cluster_count
- uint8_t sizeof_cluster_data
- uint8_t sizeof_cluster_control
- uint8_t free_clusters
- uint8_t flags

3.4.1 Detailed Description

Definition at line 113 of file rbfs.h.

3.4.2 Member Data Documentation

3.4.2.1 uint8_t rbfs_t::cluster_count

Definition at line 123 of file rbfs.h.

3.4.2.2 rbfs_memory_address_t rbfs_t::cluster_table_address

Definition at line 117 of file rbfs.h.

3.4.2.3 rbfs_driver_t rbfs_t::driver

Definition at line 114 of file rbfs.h.

3.4.2.4 uint8_t rbfs_t::flags

Definition at line 127 of file rbfs.h.

3.4.2.5 uint8_t rbfs_t::free_clusters

Definition at line 126 of file rbfs.h.

3.4.2.6 uint16_t rbfs_t::memory_size

Definition at line 115 of file rbfs.h.

3.4.2.7 uint8_t rbfs_t::resource_descriptor_count

Definition at line 122 of file rbfs.h.

3.4.2.8 rbfs_memory_address_t rbfs_t::resource_descriptor_table_address

Definition at line 116 of file rbfs.h.

3.4.2.9 uint8_t rbfs_t::sizeof_cluster

Definition at line 121 of file rbfs.h.

3.4.2.10 uint8_t rbfs_t::sizeof_cluster_control

Definition at line 125 of file rbfs.h.

3.4.2.11 uint8_t rbfs_t::sizeof_cluster_data

Definition at line 124 of file rbfs.h.

3.4.2.12 uint16_t rbfs_t::sizeof_cluster_table

Definition at line 119 of file rbfs.h.

3.4.2.13 uint8_t rbfs_t::sizeof_resource_descriptor

Definition at line 120 of file rbfs.h.

3.4.2.14 uint16_t rbfs_t::sizeof_resource_descriptor_table

Definition at line 118 of file rbfs.h.

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

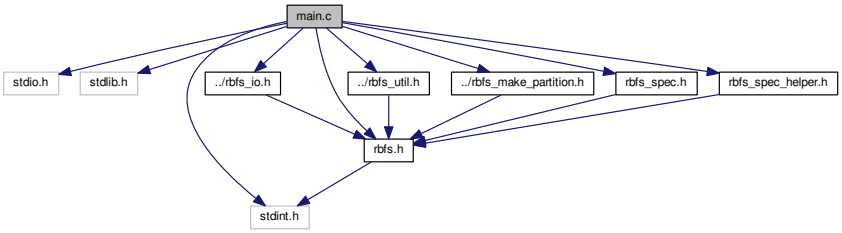
- rbfs.h

4 File Documentation

4.1 main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "../rbfs_io.h"
#include "../rbfs_util.h"
#include "../rbfs.h"
#include "../rbfs_make_partition.h"
#include "rbfs_spec.h"
#include "rbfs_spec_helper.h"
```

Include dependency graph for main.c:



Macros

- #define RBFS_SPEC_DRIVER RBFS_DRIVER_VIRTUAL

Functions

- uint8_t _rbfs_io_read (rbfs_driver_t driver, rbfs_memory_address_t address)
- void _rbfs_io_write (rbfs_driver_t driver, rbfs_memory_address_t address, uint8_t data)
- void init_rbfs_io ()
- void finish_rbfs_io ()
- int main ()

Variables

- FILE * rbfs_fp

4.1.1 Macro Definition Documentation

4.1.1.1 #define RBFS_SPEC_DRIVER RBFS_DRIVER_VIRTUAL

Definition at line 1 of file main.c.

4.1.2 Function Documentation

4.1.2.1 uint8_t _rbfs_io_read (rbfs_driver_t *driver,* rbfs_memory_address_t *address*)

rbfs - Simple Resource Based File System

rbfs_io.h

IO lib for rbfs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 71 of file main.c.

4.1.2.2 void _rbfs_io_write (rbfs_driver_t *driver*, rbfs_memory_address_t *address*, uint8_t *data*)

Definition at line 78 of file main.c.

4.1.2.3 void finish_rbfs_io ( )

Definition at line 67 of file main.c.

4.1.2.4 void init_rbfs_io ( )

Definition at line 60 of file main.c.

4.1.2.5 int main ( )

Definition at line 24 of file main.c.

4.1.3 Variable Documentation

4.1.3.1 FILE* rbfs_fp

Definition at line 16 of file main.c.

4.2 main.c

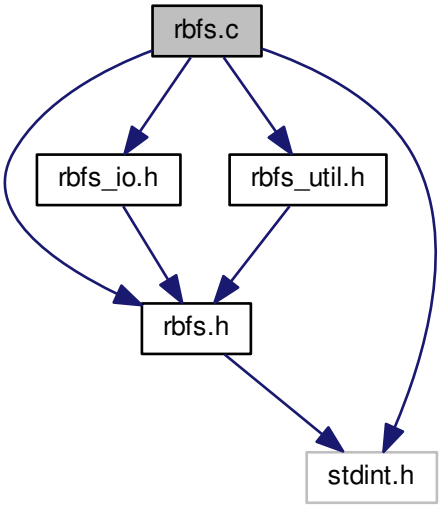
```
00001 #define RBFS_SPEC_DRIVER RBFS_DRIVER_VIRTUAL
00002
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <stdint.h>
00006
00007 #include "../rbfs_io.h"
00008 #include "../rbfs_util.h"
00009 #include "../rbfs.h"
00010 #include "../rbfs_make_partition.h"
00011
00012 #include "rbfs_spec.h"
00013 #include "rbfs_spec_helper.h"
00014
00015
00016 FILE *rbfs_fp;
00017
00018 uint8_t _rbfs_io_read(rbfs_driver_t driver,
      rbfs_memory_address_t address);
00019 void _rbfs_io_write(rbfs_driver_t driver,
      rbfs_memory_address_t address, uint8_t data);
00020
00021 void init_rbfs_io();
00022 void finish_rbfs_io();
00023
00024 int main() {
00025     rbfs_t rbfs;
00026     init_rbfs_io();
00027     rbfs_make_partition(&rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00028     format_all();
00029     rbfs_format(&rbfs);
00030     format_spec(&rbfs);
00031     mount_spec(&rbfs);
00032     umount_spec(&rbfs);
00033     alloc_resource_spec(&rbfs);
00034     try_to_alloc_resources_that_is_possible_spec(&rbfs);
00035     open_resource_spec(&rbfs);
00036     write_resource_spec(&rbfs);
00037     rewind_resource_spec(&rbfs);
00038     read_resource_spec(&rbfs);
00039     close_resource_spec(&rbfs);
00040     try_read_when_end_of_resource_is_reached_spec(&rbfs);
```

```
00041     try_read_when_resource_is_closed_spec(&rbfs);
00042     seek_resource_spec(&rbfs);
00043     random_read_resource_spec(&rbfs);
00044     random_read_with_seek_resource_spec(&rbfs);
00045     random_read_with_seek_opening_resource_spec(&rbfs);
00046     size_resource_spec(&rbfs);
00047     tell_resource_spec(&rbfs);
00048     tell_with_seek_resource_spec(&rbfs);
00049     total_space_resource_spec(&rbfs);
00050     allocating_multi_format_spec(&rbfs);
00051     read_only_mounting_spec(&rbfs);
00052     read_only_opening_spec(&rbfs);
00053
00054     rbfs_mount(RBFS_DRIVER_VIRTUAL, &rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00055     rbfs_io_memory_dump(&rbfs);
00056     finish_rbfs_io();
00057     return 0;
00058 }
00059
00060 void init_rbfs_io() {
00061     if ((rbfs_fp = fopen("img.hd", "rb+")) == NULL) {
00062         printf("Error reading img.hd");
00063         exit(1);
00064     }
00065 }
00066
00067 void finish_rbfs_io() {
00068     fclose(rbfs_fp);
00069 }
00070
00071 uint8_t _rbfs_io_read(rbfs_driver_t driver,
      rbfs_memory_address_t address) {
00072     unsigned char data;
00073     fseek(rbfs_fp, address, 0);
00074     fread(&data, sizeof(data), 1, rbfs_fp);
00075     return data;
00076 }
00077
00078 void _rbfs_io_write(rbfs_driver_t driver,
      rbfs_memory_address_t address, uint8_t data) {
00079     fseek(rbfs_fp, address, 0);
00080     fwrite(&data, sizeof(data), 1, rbfs_fp);
00081     fflush(rbfs_fp);
00082 }
```

4.3 rbfs.c File Reference

```
#include <rbfs.h>
#include <rbfs_io.h>
#include <rbfs_util.h>
#include <stdint.h>
```

Include dependency graph for rbfs.c:



Macros

- #define __RBFS_C__ 1

Functions

- rbfs_op_result_t rbfs_format (rbfs_t *rbfs)
- rbfs_op_result_t rbfs_mount (rbfs_driver_t driver, rbfs_t *rbfs, rbfs_mount_options_t options)
- rbfs_op_result_t rbfs_umount (rbfs_t *rbfs)
- rbfs_op_result_t rbfs_open (rbfs_t *rbfs, rbfs_resource_code_t resource_code, rbfs_resource_t *resource, rbfs_open_resource_options_t options)
- rbfs_op_result_t rbfs_close (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t rbfs_read (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_op_result_t rbfs_write (rbfs_t *rbfs, rbfs_resource_t *resource, uint8_t data_to_write)
- rbfs_op_result_t rbfs_seek (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_origin_t origin, rbfs_seek_int_t offset)
- rbfs_op_result_t rbfs_truncate (rbfs_t *rbfs, rbfs_resource_t *resource)
- void rbfs_sync (rbfs_t *rbfs, rbfs_resource_t *resource)
- void rbfs_stat (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_stat_t *stat)
- rbfs_op_result_t rbfs_rewind (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_resource_code_t rbfs_alloc (rbfs_t *rbfs)
- uint8_t rbfs_release (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_size (rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_tell (rbfs_resource_t *resource)
- uint8_t rbfs_eor (rbfs_resource_t *resource)
- uint8_t rbfs_error (rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_available_space (rbfs_t *rbfs)
- rbfs_resource_size_t rbfs_total_space (rbfs_t *rbfs)

Variables

- rbfs_global_flags_t rbfs_global_flags

4.3.1 Macro Definition Documentation

4.3.1.1 #define __RBFS_C__ 1

rbfs - Simple Resource Based File System

rbfs.c

A file system implementation based on the idea of resources

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file rbfs.c.

4.3.2 Function Documentation

4.3.2.1 rbfs_resource_code_t rbfs_alloc (rbfs_t * *rbfs*)

Definition at line 213 of file rbfs.c.

4.3.2.2 rbfs_resource_size_t rbfs_available_space (rbfs_t * *rbfs*)

Definition at line 271 of file rbfs.c.

4.3.2.3 rbfs_op_result_t rbfs_close (rbfs_t * *rbfs,* rbfs_resource_t * *resource*)

Definition at line 90 of file rbfs.c.

4.3.2.4 uint8_t rbfs_eor (rbfs_resource_t * *resource*)

Definition at line 263 of file rbfs.c.

4.3.2.5 uint8_t rbfs_error (rbfs_resource_t * *resource*)

Definition at line 267 of file rbfs.c.

4.3.2.6 rbfs_op_result_t rbfs_format (rbfs_t * *rbfs*)

Definition at line 21 of file rbfs.c.

4.3.2.7 rbfs_op_result_t rbfs_mount (rbfs_driver_t *driver,* rbfs_t * *rbfs,* rbfs_mount_options_t *options*)

Definition at line 33 of file rbfs.c.

4.3.2.8 rbfs_op_result_t rbfs_open (rbfs_t * *rbfs,* rbfs_resource_code_t *resource_code,* rbfs_resource_t * *resource,* rbfs_open_resource_options_t *options*)

Definition at line 54 of file rbfs.c.

4.3.2.9 uint8_t rbfs_read (rbfs_t * *rbfs,* rbfs_resource_t * *resource*)

Definition at line 97 of file rbfs.c.

4.3.2.10 uint8_t rbfs_release (rbfs_t * *rbfs,* rbfs_resource_t * *resource*)

Definition at line 241 of file rbfs.c.

4.3.2.11 rbfs_op_result_t rbfs_rewind (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 205 of file rbfs.c.

4.3.2.12 rbfs_op_result_t rbfs_seek (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_origin_t *origin*, rbfs_seek_int_t *offset*)

Definition at line 138 of file rbfs.c.

4.3.2.13 rbfs_resource_size_t rbfs_size (rbfs_resource_t * *resource*)

Definition at line 255 of file rbfs.c.

4.3.2.14 void rbfs_stat (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_stat_t * *stat*)

Definition at line 201 of file rbfs.c.

4.3.2.15 void rbfs_sync (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 192 of file rbfs.c.

4.3.2.16 rbfs_resource_size_t rbfs_tell (rbfs_resource_t * *resource*)

Definition at line 259 of file rbfs.c.

4.3.2.17 rbfs_resource_size_t rbfs_total_space (rbfs_t * *rbfs*)

Definition at line 275 of file rbfs.c.

4.3.2.18 rbfs_op_result_t rbfs_truncate (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 173 of file rbfs.c.

4.3.2.19 rbfs_op_result_t rbfs_umount (rbfs_t * *rbfs*)

Definition at line 47 of file rbfs.c.

4.3.2.20 rbfs_op_result_t rbfs_write (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, uint8_t *data_to_write*)

Definition at line 116 of file rbfs.c.

4.3.3 Variable Documentation

4.3.3.1 rbfs_global_flags_t rbfs_global_flags

Definition at line 19 of file rbfs.c.

4.4 rbfs.c

```
00001
00011 #ifndef __RBFS_C__
00012 #define __RBFS_C__ 1
00013
00014 #include <rbfs.h>
00015 #include <rbfs_io.h>
00016 #include <rbfs_util.h>
00017 #include <stdint.h>
00018
00019 rbfs_global_flags_t rbfs_global_flags;
00020
00021 rbfs_op_result_t rbfs_format(rbfs_t *rbfs) {
00022     uint8_t i;
00023     _rbfs_write_rbfs_to_disk(rbfs->driver, rbfs);
00024     for (i = 0; i < rbfs->resource_descriptor_count; i++) {
00025         _rbfs_format_resorce_descriptor(rbfs, i);
```

```
00026     }
00027     for (i = 0; i < rbfs->cluster_count; i++) {
00028         _rbfs_format_cluster(rbfs, i);
00029     }
00030     return RBFS_OP_RESULT_SUCCESS;
00031 }
00032
00033 rbfs_op_result_t rbfs_mount(rbfs_driver_t driver,
      rbfs_t *rbfs, rbfs_mount_options_t options) {
00034     if (_rbfs_is_driver_monted(driver)) {
00035         return RBFS_OP_RESULT_ERROR_DRIVER_BUSY;
00036     }
00037     _rbfs_read_rbfs_from_disk(driver, rbfs);
00038     _rbfs_set_driver_monted(driver, 1);
00039     if (options & RBFS_MOUNT_OPTION_READ_ONLY) {
00040         rbfs->flags |= RBFS_FLAG_BIT_READ_ONLY;
00041     }
00042     rbfs->driver = driver;
00043     _rbfs_free_resource_descriptors(rbfs);
00044     return RBFS_OP_RESULT_SUCCESS;
00045 }
00046
00047 rbfs_op_result_t rbfs_umount(rbfs_t *rbfs) {
00048     if (_rbfs_is_driver_monted(rbfs->driver)) {
00049         _rbfs_set_driver_monted(rbfs->driver, 0);
00050     }
00051     return RBFS_OP_RESULT_SUCCESS;
00052 }
00053
00054 rbfs_op_result_t rbfs_open(rbfs_t *rbfs,
      rbfs_resource_code_t resource_code, rbfs_resource_t *resource,
      rbfs_open_resource_options_t options) {
00055     uint8_t i;
00056     rbfs_memory_address_t address;
00057     rbfs_resource_descriptor_t resource_descriptor;
00058     uint8_t flags;
00059     if (!_rbfs_is_driver_monted(rbfs->driver)) {
00060         return RBFS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED;
00061     }
00062     resource_descriptor = _rbfs_resource_code_to_resource_descriptor
      (resource_code);
00063     address = _rbfs_resource_descriptor_to_address(rbfs,
      resource_descriptor);
00064     flags = _rbfs_io_read(rbfs->driver, RD_ADDRESS_TO_FLAG(address));
00065     if (!(flags & RBFS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00066         return RBFS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED
      ;
00067     }
00068     if (flags & RBFS_RESOURCE_FLAG_BIT_OPENED) {
00069         return RBFS_OP_RESULT_ERROR_RESOURCE_OPENED;
00070     }
00071     flags |= RBFS_RESOURCE_FLAG_BIT_OPENED;
00072     if ((options & RBFS_OPEN_RESOURCE_OPTION_READ_ONLY) || (rbfs->
      flags & RBFS_FLAG_BIT_READ_ONLY)) {
00073         flags |= RBFS_RESOURCE_FLAG_BIT_READ_ONLY;
00074
00075     }
00076     _rbfs_io_write(rbfs->driver, RD_ADDRESS_TO_FLAG(address), flags);
00077     resource->resource_descriptor = resource_descriptor;
00078     resource->first_cluster = _rbfs_io_read(rbfs->
      driver, RD_ADDRESS_TO_FIRST_CLUSTER(address));
00079     resource->current_cluster = resource->first_cluster;
00080     resource->cluster_offset = rbfs->sizeof_cluster_control;
00081     resource->current_position = 0;
00082     for (i = 0; i < RBFS_SIZEOF_RESOURCE_SIZE; i++) {
00083         *((uint8_t *) (&resource->size) + i) = _rbfs_io_read(rbfs->
      driver, address + i);
00084     }
00085     resource->flags = flags;
00086     _rbfs_check_for_eor_reached(resource);
00087     return RBFS_OP_RESULT_SUCCESS;
00088 }
00089
00090 rbfs_op_result_t rbfs_close(rbfs_t *rbfs,
      rbfs_resource_t *resource) {
00091     rbfs_sync(rbfs, resource);
00092     _rbfs_free_resource_descriptor(rbfs, resource->
      resource_descriptor);
00093     resource->flags = ~RBFS_RESOURCE_FLAG_BIT_OPENED;
00094     return RBFS_OP_RESULT_SUCCESS;
00095 }
00096
00097 uint8_t rbfs_read(rbfs_t *rbfs, rbfs_resource_t *resource) {
00098     rbfs_memory_address_t address;
00099     uint8_t read_data;
00100     if (!(resource->flags & RBFS_RESOURCE_FLAG_BIT_OPENED)) {
00101         resource->flags |= RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
```

```
    ;
00102         return 0;
00103     }
00104     if (_rbfs_is_eor_reached(resource)) {
00105         return 0;
00106     }
00107     _rbfs_check_for_availability(rbfs, resource);
00108     address = _rbfs_cluster_to_address(rbfs, resource->
    current_cluster);
00109     read_data = _rbfs_io_read(rbfs->driver, address + resource->
    cluster_offset);
00110     resource->current_position++;
00111     resource->cluster_offset++;
00112     _rbfs_check_for_eor_reached(resource);
00113     return read_data;
00114 }
00115
00116 rbfs_op_result_t rbfs_write(rbfs_t *rbfs,
    rbfs_resource_t *resource, uint8_t data_to_write) {
00117     rbfs_memory_address_t address;
00118     if (!(resource->flags & RBFS_RESOURCE_FLAG_BIT_OPENED)) {
00119         return RBFS_OP_RESULT_ERROR_RESOURCE_CLOSED;
00120     }
00121     if (resource->flags & RBFS_RESOURCE_FLAG_BIT_READ_ONLY) {
00122         return RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY;
00123     }
00124     if (!_rbfs_check_for_availability(rbfs, resource)) {
00125         return RBFS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE;
00126     }
00127     address = _rbfs_cluster_to_address(rbfs, resource->
    current_cluster);
00128     _rbfs_io_write(rbfs->driver, address + resource->
    cluster_offset, data_to_write);
00129     resource->cluster_offset++;
00130     resource->current_position++;
00131     if (rbfs_eor(resource)) {
00132         resource->size++;
00133         rbfs_sync(rbfs, resource);
00134     }
00135     return RBFS_OP_RESULT_SUCCESS;
00136 }
00137
00138 rbfs_op_result_t rbfs_seek(rbfs_t *rbfs,
    rbfs_resource_t *resource, rbfs_seek_origin_t origin,
    rbfs_seek_int_t offset) {
00139     int16_t new_position = 0;
00140     if (resource->size == 0) {
00141         return RBFS_OP_RESULT_SUCCESS;
00142     }
00143     switch (origin) {
00144         case RBFS_SEEK_ORIGIN_BEGIN:
00145             new_position = offset;
00146             break;
00147         case RBFS_SEEK_ORIGIN_CURRENT:
00148             new_position = resource->current_position + offset;
00149             break;
00150     }
00151     new_position %= resource->size + 1;
00152     if (new_position < 0) {
00153         new_position += resource->size;
00154     }
00155     if (new_position == 0) {
00156         rbfs_rewind(rbfs, resource);
00157         return RBFS_OP_RESULT_SUCCESS;
00158     }
00159     if (new_position < resource->current_position) {
00160         if (new_position > (resource->current_position - new_position)) {
00161             _rbfs_move_current_position_back(rbfs, resource, (resource->
    current_position - new_position));
00162         } else {
00163             rbfs_rewind(rbfs, resource);
00164             _rbfs_move_current_position_ahead(rbfs, resource, new_position
    );
00165         }
00166     } else {
00167         _rbfs_move_current_position_ahead(rbfs, resource, (new_position -
    resource->current_position));
00168     }
00169     _rbfs_check_for_eor_reached(resource);
00170     return RBFS_OP_RESULT_SUCCESS;
00171 }
00172
00173 rbfs_op_result_t rbfs_truncate(rbfs_t *rbfs,
    rbfs_resource_t *resource) {
00174     uint8_t flags;
00175     rbfs_memory_address_t resource_descriptor_address;
00176     uint8_t freed_clusters = 0;
```

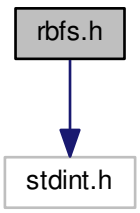
```
00177     resource_descriptor_address = _rbfs_resource_descriptor_to_address(
      rbfs, resource->resource_descriptor);
00178     flags = _rbfs_io_read(rbfs->driver, RD_ADDRESS_TO_FLAG(
      resource_descriptor_address));
00179     if (!(flags & RBFS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00180         return RBFS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED
      ;
00181     }
00182     if (resource->size > rbfs->sizeof_cluster_data) {
00183         freed_clusters = _rbfs_format_clusterbfs_chain(rbfs,
      _rbfs_next_cluster_by_cluster(rbfs, resource->
      first_cluster));
00184     }
00185     _rbfs_increase_free_clusters(rbfs, freed_clusters);
00186     resource->size = 0x00;
00187     _rbfs_io_write(rbfs->driver, RD_ADDRESS_TO_SIZE_LOW(
      resource_descriptor_address), 0x00);
00188     _rbfs_io_write(rbfs->driver, RD_ADDRESS_TO_SIZE_HIGH(
      resource_descriptor_address), 0x00);
00189     return RBFS_OP_RESULT_SUCCESS;
00190 }
00191
00192 void rbfs_sync(rbfs_t *rbfs, rbfs_resource_t *resource) {
00193     uint8_t i;
00194     rbfs_memory_address_t address;
00195     address = _rbfs_resource_descriptor_to_address(rbfs, resource->
      resource_descriptor);
00196     for (i = 0; i < 2; i++) {
00197         _rbfs_io_write(rbfs->driver, address + i, *((uint8_t *) (&(resource->
      size)) + i));
00198     }
00199 }
00200
00201 void rbfs_stat(rbfs_t *rbfs, rbfs_resource_t *resource,
      rbfs_stat_t *stat) { // TODO
00202     stat->flags = 0xff;
00203 }
00204
00205 rbfs_op_result_t rbfs_rewind(rbfs_t *rbfs,
      rbfs_resource_t *resource) {
00206     resource->current_cluster = resource->first_cluster;
00207     resource->cluster_offset = rbfs->sizeof_cluster_control;
00208     resource->current_position = 0;
00209     _rbfs_check_for_eor_reached(resource);
00210     return RBFS_OP_RESULT_SUCCESS;
00211 }
00212
00213 rbfs_resource_code_t rbfs_alloc(rbfs_t *rbfs) {
00214     uint8_t i;
00215     uint8_t flags;
00216     rbfs_cluster_t first_cluster;
00217     rbfs_memory_address_t resource_descriptor_address, cluster_address;
00218     if (rbfs->free_clusters < 1) {
00219         return RBFS_NULL_RESOURCE_CODE;
00220     }
00221     resource_descriptor_address = rbfs->resource_descriptor_table_address;
00222     for (i = 0; i < rbfs->resource_descriptor_count; i++) {
00223         flags = _rbfs_io_read(rbfs->driver,
      RD_ADDRESS_TO_FLAG(resource_descriptor_address));
00224         if (!(flags & RBFS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00225             cluster_address = _rbfs_alloc_cluster(rbfs);
00226             if (cluster_address == RBFS_NULL_CLUSTER_ADDRESS) {
00227                 return RBFS_NULL_RESOURCE_CODE;
00228             }
00229             flags |= RBFS_RESOURCE_FLAG_BIT_ALLOCATED;
00230             first_cluster = _rbfs_address_to_cluster(rbfs, cluster_address);
00231             _rbfs_create_cluster_chain(rbfs, first_cluster,
      RBFS_INEXISTENT_CLUSTER);
00232             _rbfs_io_write(rbfs->driver,
      RD_ADDRESS_TO_FIRST_CLUSTER(resource_descriptor_address), first_cluster);
00233             _rbfs_io_write(rbfs->driver, RD_ADDRESS_TO_FLAG(
      resource_descriptor_address), flags);
00234             return _rbfs_resource_descriptor_to_resource_code(i);
00235         }
00236         resource_descriptor_address += rbfs->sizeof_resource_descriptor;
00237     }
00238     return RBFS_NULL_RESOURCE_CODE;
00239 }
00240
00241 uint8_t rbfs_release(rbfs_t *rbfs, rbfs_resource_t *resource) {
00242     uint8_t flags;
00243     rbfs_memory_address_t resource_descriptor_address;
00244     resource_descriptor_address = _rbfs_resource_descriptor_to_address(
      rbfs, resource->resource_descriptor);
00245     flags = _rbfs_io_read(rbfs->driver, RD_ADDRESS_TO_FLAG(
      resource_descriptor_address));
00246     if (!(flags & RBFS_RESOURCE_FLAG_BIT_ALLOCATED)) {
```

```
00247         return 1;
00248     }
00249     _rbfs_format_resource_clusters(rbfs, resource);
00250     _rbfs_format_resorce_descriptor(rbfs, resource->
      resource_descriptor);
00251     resource->flags = 0x00;
00252     return 1;
00253 }
00254
00255 rbfs_resource_size_t rbfs_size(rbfs_resource_t *resource) {
00256     return resource->size;
00257 }
00258
00259 rbfs_resource_size_t rbfs_tell(rbfs_resource_t *resource) {
00260     return resource->current_position;
00261 }
00262
00263 uint8_t rbfs_eor(rbfs_resource_t *resource) {
00264     return _rbfs_is_eor_reached(resource);
00265 }
00266
00267 uint8_t rbfs_error(rbfs_resource_t *resource) {
00268     return (resource->flags & RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
       || resource->flags & RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE
      );
00269 }
00270
00271 rbfs_resource_size_t rbfs_available_space(
      rbfs_t *rbfs) {
00272     return rbfs->free_clusters * rbfs->sizeof_cluster_data;
00273 }
00274
00275 rbfs_resource_size_t rbfs_total_space(rbfs_t *rbfs) {
00276     return rbfs->cluster_count * rbfs->sizeof_cluster_data;
00277 }
00278
00279 #endif // __RBFS_C__
```

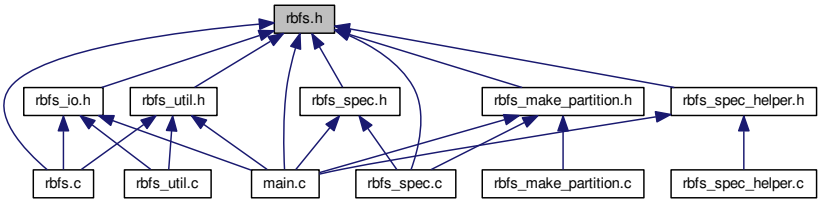
4.5 rbfs.h File Reference

`#include <stdint.h>`

Include dependency graph for rbfs.h:



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



Classes

- struct rbfs_stat_t
- struct rbfs_t
- struct rbfs_resource_t
- struct rbfs_global_flags_t

Macros

- #define RBFS_NULL_RESOURCE_CODE 0xff
- #define RBFS_NULL_CLUSTER 0xff
- #define RBFS_NULL_RESORCE_DESCRIPTOR_ADDRESS 0xff
- #define RBFS_NULL_CLUSTER_ADDRESS 0x00
- #define RBFS_FIRST_ADDRESS_OF_MEMORY 0x00
- #define RBFS_SIZEOF_RESOURCE_SIZE 0x02
- #define RBFS_INEXISTENT_CLUSTER 0xff
- #define CLUSTER_ADDRESS_TO_NEXT(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 0)
- #define CLUSTER_ADDRESS_TO_PREV(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 1)
- #define CLUSTER_ADDRESS_TO_DATA(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 2)
- #define RD_ADDRESS_TO_SIZE_LOW(RD_ADDRESS) ((RD_ADDRESS) + 0)
- #define RD_ADDRESS_TO_SIZE_HIGH(RD_ADDRESS) ((RD_ADDRESS) + 1)
- #define RD_ADDRESS_TO_FIRST_CLUSTER(RD_ADDRESS) ((RD_ADDRESS) + 2)
- #define RD_ADDRESS_TO_FLAG(RD_ADDRESS) ((RD_ADDRESS) + 3)

Typedefs

- typedef uint8_t rbfs_resource_descriptor_t
- typedef uint8_t rbfs_cluster_t
- typedef uint16_t rbfs_resource_size_t
- typedef uint16_t rbfs_memory_address_t
- typedef uint8_t rbfs_resource_code_t
- typedef uint16_t rbfs_seek_int_t

Enumerations

- enum rbfs_driver_t {
  RBFS_DRIVER_VIRTUAL = 0, RBFS_DRIVER_SELF_EEPROM = 1, RBFS_DRIVER_MULTI_EXTERN↩
  AL_EEPROM = 2, RBFS_DRIVER_EXTERNAL_EEPROM = 3,
  RBFS_DRIVER_ARDUINO_EEPROM = 4 }

- enum rbfs_resource_flag_bits_t { RBFS_RESOURCE_FLAG_BIT_OPENED = 1, RBFS_RESOURCE_FLAG_BIT_READ_ONLY = 2, RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ = 4, RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE = 8, RBFS_RESOURCE_FLAG_BIT_ALLOCATED = 16, RBFS_RESOURCE_FLAG_BIT_EOR_REACHED = 32 }
- enum rbfs_open_resource_options_t { RBFS_OPEN_RESOURCE_OPTION_NORMAL = 0, RBFS_OPEN_RESOURCE_OPTION_READ_ONLY = 1 }
- enum rbfs_mount_options_t { RBFS_MOUNT_OPTION_NORMAL = 0, RBFS_MOUNT_OPTION_READ_ONLY = 1 }
- enum rbfs_flag_bits_t { RBFS_FLAG_BIT_DRIVER_MOUNTED = 1, RBFS_FLAG_BIT_READ_ONLY = 2 }
- enum rbfs_op_result_t { RBFS_OP_RESULT_SUCCESS = 0, RBFS_OP_RESULT_ERROR_RESOURCE_OPENED = 1, RBFS_OP_RESULT_ERROR_RESOURCE_CLOSED = 2, RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY = 3, RBFS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE = 4, RBFS_OP_RESULT_ERROR_DRIVER_BUSY = 5, RBFS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND = 6, RBFS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED = 7, RBFS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED = 8 }
- enum rbfs_seek_origin_t { RBFS_SEEK_ORIGIN_BEGIN = 0, RBFS_SEEK_ORIGIN_CURRENT = 1 }

Functions

- rbfs_op_result_t rbfs_format (rbfs_t *rbfs)
- rbfs_op_result_t rbfs_mount (rbfs_driver_t driver, rbfs_t *rbfs, rbfs_mount_options_t options)
- rbfs_op_result_t rbfs_umount (rbfs_t *rbfs)
- rbfs_op_result_t rbfs_open (rbfs_t *rbfs, rbfs_resource_code_t resource_code, rbfs_resource_t *resource, rbfs_open_resource_options_t options)
- rbfs_op_result_t rbfs_close (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t rbfs_read (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_op_result_t rbfs_write (rbfs_t *rbfs, rbfs_resource_t *resource, uint8_t data_to_write)
- rbfs_op_result_t rbfs_seek (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_origin_t origin, rbfs_seek_int_t offset)
- rbfs_op_result_t rbfs_truncate (rbfs_t *rbfs, rbfs_resource_t *resource)
- void rbfs_sync (rbfs_t *rbfs, rbfs_resource_t *resource)
- void rbfs_stat (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_stat_t *stat)
- rbfs_op_result_t rbfs_rewind (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_resource_code_t rbfs_alloc (rbfs_t *rbfs)
- uint8_t rbfs_release (rbfs_t *rbfs, rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_size (rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_tell (rbfs_resource_t *resource)
- uint8_t rbfs_eor (rbfs_resource_t *resource)
- uint8_t rbfs_error (rbfs_resource_t *resource)
- rbfs_resource_size_t rbfs_available_space (rbfs_t *rbfs)
- rbfs_resource_size_t rbfs_total_space (rbfs_t *rbfs)

Variables

- rbfs_global_flags_t rbfs_global_flags

4.5.1 Macro Definition Documentation

4.5.1.1 #define CLUSTER_ADDRESS_TO_DATA(*CLUSTER_ADDRESS*) ((CLUSTER_ADDRESS) + 2)

Definition at line 30 of file rbfs.h.

4.5.1.2 #define CLUSTER_ADDRESS_TO_NEXT(*CLUSTER_ADDRESS*) ((CLUSTER_ADDRESS) + 0)

Definition at line 28 of file rbfs.h.

4.5.1.3 #define CLUSTER_ADDRESS_TO_PREV(*CLUSTER_ADDRESS*) ((CLUSTER_ADDRESS) + 1)

Definition at line 29 of file rbfs.h.

4.5.1.4 #define RBFS_FIRST_ADDRESS_OF_MEMORY 0x00

Definition at line 22 of file rbfs.h.

4.5.1.5 #define RBFS_INEXISTENT_CLUSTER 0xff

Definition at line 26 of file rbfs.h.

4.5.1.6 #define RBFS_NULL_CLUSTER 0xff

Definition at line 17 of file rbfs.h.

4.5.1.7 #define RBFS_NULL_CLUSTER_ADDRESS 0x00

Definition at line 20 of file rbfs.h.

4.5.1.8 #define RBFS_NULL_RESORCE_DESCRIPTOR_ADDRESS 0xff

Definition at line 19 of file rbfs.h.

4.5.1.9 #define RBFS_NULL_RESOURCE_CODE 0xff

rbfs - Simple Resource Based File System

rbfs.h

An file system header definition based on the idea of resources

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 16 of file rbfs.h.

4.5.1.10 #define RBFS_SIZEOF_RESOURCE_SIZE 0x02

Definition at line 24 of file rbfs.h.

4.5.1.11 #define RD_ADDRESS_TO_FIRST_CLUSTER(*RD_ADDRESS*) ((RD_ADDRESS) + 2)

Definition at line 34 of file rbfs.h.

4.5.1.12 #define RD_ADDRESS_TO_FLAG(*RD_ADDRESS*) ((RD_ADDRESS) + 3)

Definition at line 35 of file rbfs.h.

4.5.1.13 #define RD_ADDRESS_TO_SIZE_HIGH(*RD_ADDRESS*) ((RD_ADDRESS) + 1)

Definition at line 33 of file rbfs.h.

4.5.1.14 #define RD_ADDRESS_TO_SIZE_LOW(*RD_ADDRESS*) ((RD_ADDRESS) + 0)

Definition at line 32 of file rbfs.h.

4.5.2 Typedef Documentation

4.5.2.1 typedef uint8_t rbfs_cluster_t

Definition at line 38 of file rbfs.h.

4.5.2.2 typedef uint16_t rbfs_memory_address_t

Definition at line 40 of file rbfs.h.

4.5.2.3 typedef uint8_t rbfs_resource_code_t

Definition at line 41 of file rbfs.h.

4.5.2.4 typedef uint8_t rbfs_resource_descriptor_t

Definition at line 37 of file rbfs.h.

4.5.2.5 typedef uint16_t rbfs_resource_size_t

Definition at line 39 of file rbfs.h.

4.5.2.6 typedef uint16_t rbfs_seek_int_t

Definition at line 42 of file rbfs.h.

4.5.3 Enumeration Type Documentation

4.5.3.1 enum rbfs_driver_t

Enumerator

RBFS_DRIVER_VIRTUAL

RBFS_DRIVER_SELF_EEPROM

RBFS_DRIVER_MULTI_EXTERNAL_EEPROM

RBFS_DRIVER_EXTERNAL_EEPROM

RBFS_DRIVER_ARDUINO_EEPROM

Definition at line 46 of file rbfs.h.

4.5.3.2 enum rbfs_flag_bits_t

Enumerator

RBFS_FLAG_BIT_DRIVER_MOUNTED

RBFS_FLAG_BIT_READ_ONLY

Definition at line 81 of file rbfs.h.

4.5.3.3 enum rbfs_mount_options_t

Enumerator

RBFS_MOUNT_OPTION_NORMAL

RBFS_MOUNT_OPTION_READ_ONLY

Definition at line 74 of file rbfs.h.

4.5.3.4 enum rbfs_op_result_t

Enumerator

RBFS_OP_RESULT_SUCCESS

RBFS_OP_RESULT_ERROR_RESOURCE_OPENED

RBFS_OP_RESULT_ERROR_RESOURCE_CLOSED

RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY

RBFS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE

RBFS_OP_RESULT_ERROR_DRIVER_BUSY

RBFS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND

RBFS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED

RBFS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED

Definition at line 88 of file rbfs.h.

4.5.3.5 enum rbfs_open_resource_options_t

Enumerator

RBFS_OPEN_RESOURCE_OPTION_NORMAL

RBFS_OPEN_RESOURCE_OPTION_READ_ONLY

Definition at line 67 of file rbfs.h.

4.5.3.6 enum rbfs_resource_flag_bits_t

Enumerator

RBFS_RESOURCE_FLAG_BIT_OPENED

RBFS_RESOURCE_FLAG_BIT_READ_ONLY

RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ

RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE

RBFS_RESOURCE_FLAG_BIT_ALLOCATED

RBFS_RESOURCE_FLAG_BIT_EOR_REACHED

Definition at line 56 of file rbfs.h.

4.5.3.7 enum rbfs_seek_origin_t

Enumerator

RBFS_SEEK_ORIGIN_BEGIN

RBFS_SEEK_ORIGIN_CURRENT

Definition at line 102 of file rbfs.h.

4.5.4 Function Documentation

4.5.4.1 rbfs_resource_code_t rbfs_alloc (rbfs_t * *rbfs*)

Definition at line 213 of file rbfs.c.

4.5.4.2 rbfs_resource_size_t rbfs_available_space (rbfs_t * *rbfs*)

Definition at line 271 of file rbfs.c.

4.5.4.3 rbfs_op_result_t rbfs_close (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 90 of file rbfs.c.

4.5.4.4 uint8_t rbfs_eor (rbfs_resource_t * *resource*)

Definition at line 263 of file rbfs.c.

4.5.4.5 uint8_t rbfs_error (rbfs_resource_t * *resource*)

Definition at line 267 of file rbfs.c.

4.5.4.6 rbfs_op_result_t rbfs_format (rbfs_t * *rbfs*)

Definition at line 21 of file rbfs.c.

4.5.4.7 rbfs_op_result_t rbfs_mount (rbfs_driver_t *driver*, rbfs_t * *rbfs*, rbfs_mount_options_t *options*)

Definition at line 33 of file rbfs.c.

4.5.4.8 rbfs_op_result_t rbfs_open (rbfs_t * *rbfs*, rbfs_resource_code_t *resource_code*, rbfs_resource_t * *resource*, rbfs_open_resource_options_t *options*)

Definition at line 54 of file rbfs.c.

4.5.4.9 uint8_t rbfs_read (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 97 of file rbfs.c.

4.5.4.10 uint8_t rbfs_release (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 241 of file rbfs.c.

4.5.4.11 rbfs_op_result_t rbfs_rewind (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 205 of file rbfs.c.

4.5.4.12 rbfs_op_result_t rbfs_seek (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_origin_t *origin*, rbfs_seek_int_t *offset*)

Definition at line 138 of file rbfs.c.

4.5.4.13 rbfs_resource_size_t rbfs_size (rbfs_resource_t * *resource*)

Definition at line 255 of file rbfs.c.

4.5.4.14 void rbfs_stat (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_stat_t * *stat*)

Definition at line 201 of file rbfs.c.

4.5.4.15 void rbfs_sync (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 192 of file rbfs.c.

4.5.4.16 rbfs_resource_size_t rbfs_tell (rbfs_resource_t * *resource*)

Definition at line 259 of file rbfs.c.

4.5.4.17 rbfs_resource_size_t rbfs_total_space (rbfs_t * *rbfs*)

Definition at line 275 of file rbfs.c.

4.5.4.18 rbfs_op_result_t rbfs_truncate (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Definition at line 173 of file rbfs.c.

4.5.4.19 rbfs_op_result_t rbfs_umount (rbfs_t * *rbfs*)

Definition at line 47 of file rbfs.c.

4.5.4.20 rbfs_op_result_t rbfs_write (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, uint8_t *data_to_write*)

Definition at line 116 of file rbfs.c.

4.5.5 Variable Documentation

4.5.5.1 rbfs_global_flags_t rbfs_global_flags

Definition at line 19 of file rbfs.c.

4.6 rbfs.h

```
00001
00011 #ifndef __RBFS_H__
00012 #define __RBFS_H__ 1
00013
00014 #include <stdint.h>
00015
00016 #define RBFS_NULL_RESOURCE_CODE                                  0xff
00017 #define RBFS_NULL_CLUSTER                                        0xff
00018
00019 #define RBFS_NULL_RESORCE_DESCRIPTOR_ADDRESS                     0xff
00020 #define RBFS_NULL_CLUSTER_ADDRESS                                0x00
00021
00022 #define RBFS_FIRST_ADDRESS_OF_MEMORY                             0x00
00023
00024 #define RBFS_SIZEOF_RESOURCE_SIZE                                0x02
00025
00026 #define RBFS_INEXISTENT_CLUSTER                                  0xff
00027
00028 #define CLUSTER_ADDRESS_TO_NEXT(CLUSTER_ADDRESS)                  ((CLUSTER_ADDRESS) + 0)
00029 #define CLUSTER_ADDRESS_TO_PREV(CLUSTER_ADDRESS)                  ((CLUSTER_ADDRESS) + 1)
00030 #define CLUSTER_ADDRESS_TO_DATA(CLUSTER_ADDRESS)                  ((CLUSTER_ADDRESS) + 2)
00031
00032 #define RD_ADDRESS_TO_SIZE_LOW(RD_ADDRESS)                        ((RD_ADDRESS) + 0)
00033 #define RD_ADDRESS_TO_SIZE_HIGH(RD_ADDRESS)                       ((RD_ADDRESS) + 1)
00034 #define RD_ADDRESS_TO_FIRST_CLUSTER(RD_ADDRESS)                   ((RD_ADDRESS) + 2)
00035 #define RD_ADDRESS_TO_FLAG(RD_ADDRESS)                            ((RD_ADDRESS) + 3)
00036
00037 typedef uint8_t  rbfs_resource_descriptor_t;
00038 typedef uint8_t  rbfs_cluster_t;
00039 typedef uint16_t rbfs_resource_size_t;
00040 typedef uint16_t rbfs_memory_address_t;
00041 typedef uint8_t  rbfs_resource_code_t;
00042 typedef uint16_t rbfs_seek_int_t;
00043
00044 // Drivers
00045
00046 typedef enum {
00047     RBFS_DRIVER_VIRTUAL = 0,
00048     RBFS_DRIVER_SELF_EEPROM = 1,
00049     RBFS_DRIVER_MULTI_EXTERNAL_EEPROM = 2,
00050     RBFS_DRIVER_EXTERNAL_EEPROM = 3,
00051     RBFS_DRIVER_ARDUINO_EEPROM = 4
00052 } rbfs_driver_t;
00053
00054 // Resource fag bit values
00055
00056 typedef enum {
00057     RBFS_RESOURCE_FLAG_BIT_OPENED = 1,
00058     RBFS_RESOURCE_FLAG_BIT_READ_ONLY = 2,
00059     RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ = 4,
00060     RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE = 8,
00061     RBFS_RESOURCE_FLAG_BIT_ALLOCATED = 16,
00062     RBFS_RESOURCE_FLAG_BIT_EOR_REACHED = 32
00063 } rbfs_resource_flag_bits_t;
00064
00065 // Options to open a resource
```

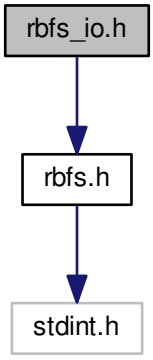
```
00066
00067 typedef enum {
00068     RBFS_OPEN_RESOURCE_OPTION_NORMAL = 0,
00069     RBFS_OPEN_RESOURCE_OPTION_READ_ONLY = 1
00070 } rbfs_open_resource_options_t;
00071
00072 // Options to mount a resource
00073
00074 typedef enum {
00075     RBFS_MOUNT_OPTION_NORMAL = 0,
00076     RBFS_MOUNT_OPTION_READ_ONLY = 1
00077 } rbfs_mount_options_t;
00078
00079 // Rs fag bit values
00080
00081 typedef enum {
00082     RBFS_FLAG_BIT_DRIVER_MOUNTED = 1,
00083     RBFS_FLAG_BIT_READ_ONLY = 2
00084 } rbfs_flag_bits_t;
00085
00086 // Operation result
00087
00088 typedef enum {
00089     RBFS_OP_RESULT_SUCCESS = 0,
00090     RBFS_OP_RESULT_ERROR_RESOURCE_OPENED = 1,
00091     RBFS_OP_RESULT_ERROR_RESOURCE_CLOSED = 2,
00092     RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY = 3,
00093     RBFS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE = 4,
00094     RBFS_OP_RESULT_ERROR_DRIVER_BUSY = 5,
00095     RBFS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND = 6,
00096     RBFS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED = 7,
00097     RBFS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED = 8
00098 } rbfs_op_result_t;
00099
00100 // Seek position reference
00101
00102 typedef enum {
00103     RBFS_SEEK_ORIGIN_BEGIN = 0,
00104     RBFS_SEEK_ORIGIN_CURRENT = 1
00105 } rbfs_seek_origin_t;
00106
00107 typedef struct {
00108     uint8_t flags;
00109 } rbfs_stat_t;
00110
00111 // Resource system
00112
00113 typedef struct {
00114     rbfs_driver_t driver;
00115     uint16_t memory_size;
00116     rbfs_memory_address_t resource_descriptor_table_address
      ;
00117     rbfs_memory_address_t cluster_table_address;
00118     uint16_t sizeof_resource_descriptor_table;
00119     uint16_t sizeof_cluster_table;
00120     uint8_t sizeof_resource_descriptor;
00121     uint8_t sizeof_cluster;
00122     uint8_t resource_descriptor_count;
00123     uint8_t cluster_count;
00124     uint8_t sizeof_cluster_data;
00125     uint8_t sizeof_cluster_control;
00126     uint8_t free_clusters;
00127     uint8_t flags;
00128 } rbfs_t;
00129
00130 // Resource
00131
00132 typedef struct {
00133     rbfs_resource_descriptor_t resource_descriptor;
00134     rbfs_cluster_t first_cluster;
00135     rbfs_cluster_t current_cluster;
00136     uint8_t cluster_offset;
00137     uint16_t size;
00138     uint16_t current_position;
00139     uint8_t flags;
00140 } rbfs_resource_t;
00141
00142 typedef struct {
00143     uint8_t driver_mouted;
00144 } rbfs_global_flags_t;
00145
00146 extern rbfs_global_flags_t rbfs_global_flags;
00147
00148 // Format a device
00149 rbfs_op_result_t rbfs_format(rbfs_t *rbfs);
00150
00151 // Register a work area
```

```
00152 rbfs_op_result_t rbfs_mount(rbfs_driver_t driver,
      rbfs_t *rbfs, rbfs_mount_options_t options);
00153
00154 // Unregister a work area
00155 rbfs_op_result_t rbfs_umount(rbfs_t *rbfs);
00156
00157 // Open/Create a resource (you must give a empty resource)
00158 rbfs_op_result_t rbfs_open(rbfs_t *rbfs,
      rbfs_resource_code_t resource_code, rbfs_resource_t *resource,
      rbfs_open_resource_options_t options);
00159
00160 // Close a resource
00161 rbfs_op_result_t rbfs_close(rbfs_t *rbfs,
      rbfs_resource_t *resource);
00162
00163 // Read a byte from resource
00164 uint8_t rbfs_read(rbfs_t *rbfs, rbfs_resource_t *resource);
00165
00166 // Write a byte from resource
00167 rbfs_op_result_t rbfs_write(rbfs_t *rbfs,
      rbfs_resource_t *resource, uint8_t data_to_write);
00168
00169 // Move read/write pointer, (Expand resource size not implemented yet)
00170 rbfs_op_result_t rbfs_seek(rbfs_t *rbfs,
      rbfs_resource_t *resource, rbfs_seek_origin_t origin,
      rbfs_seek_int_t offset);
00171
00172 // Truncate resource size
00173 rbfs_op_result_t rbfs_truncate(rbfs_t *rbfs,
      rbfs_resource_t *resource);
00174
00175 // Flush cached data
00176 void rbfs_sync(rbfs_t *rbfs, rbfs_resource_t *resource);
00177
00178 // Get descriptor status
00179 void rbfs_stat(rbfs_t *rbfs, rbfs_resource_t *resource,
      rbfs_stat_t *stat);
00180
00181 // Rewind the position of a resource pointer
00182 rbfs_op_result_t rbfs_rewind(rbfs_t *rbfs,
      rbfs_resource_t *resource);
00183
00184 // Create/Allocate a new resource if available
00185 rbfs_resource_code_t rbfs_alloc(rbfs_t *rbfs);
00186
00187 // Make a resource free to be allocated for another one
00188 uint8_t rbfs_release(rbfs_t *rbfs, rbfs_resource_t *resource);
00189
00190 // Get size of a resource
00191 rbfs_resource_size_t rbfs_size(rbfs_resource_t *resource);
00192
00193 // Get the current read/write pointer
00194 rbfs_resource_size_t rbfs_tell(rbfs_resource_t *resource);
00195
00196 // Test for end-of-resource on a resource
00197 uint8_t rbfs_eor(rbfs_resource_t *resource);
00198
00199 // Test for an error on a resource
00200 uint8_t rbfs_error(rbfs_resource_t *resource);
00201
00202 // Return the current available space in the partition
00203 rbfs_resource_size_t rbfs_available_space(
      rbfs_t *rbfs);
00204
00205 // Return the total space in the partition
00206 rbfs_resource_size_t rbfs_total_space(rbfs_t *rbfs);
00207
00208 #endif // __RBFS_H__
```

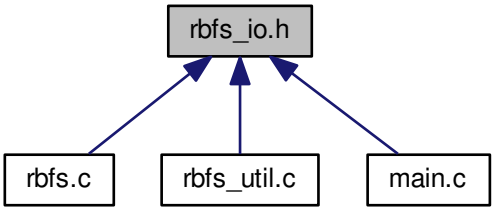
4.7 rbfs_io.h File Reference

```
#include "rbfs.h"
```

Include dependency graph for rbfs_io.h:



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



Functions

- uint8_t _rbfs_io_read (rbfs_driver_t driver, rbfs_memory_address_t address)
- void _rbfs_io_write (rbfs_driver_t driver, rbfs_memory_address_t address, uint8_t data)

4.7.1 Function Documentation

4.7.1.1 uint8_t _rbfs_io_read (**rbfs_driver_t** *driver,* **rbfs_memory_address_t** *address*)

rbfs - Simple Resource Based File System

rbfs_io.h

IO lib for rbfs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 71 of file main.c.

4.7.1.2 void _rbfs_io_write (rbfs_driver_t *driver*, rbfs_memory_address_t *address*, uint8_t *data*)

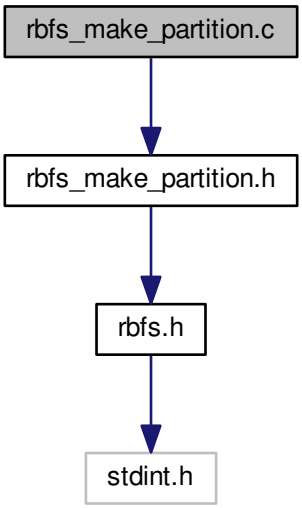
Definition at line 78 of file main.c.

4.8 rbfs_io.h

```
00001
00011 #ifndef __RBFS_IO_H__
00012 #define __RBFS_IO_H__ 1
00013
00014 #include "rbfs.h"
00015
00016 extern uint8_t _rbfs_io_read(rbfs_driver_t driver,
      rbfs_memory_address_t address);
00017
00018 extern void _rbfs_io_write(rbfs_driver_t driver,
      rbfs_memory_address_t address, uint8_t data);
00019
00020 #endif // __RBFS_IO_H__
```

4.9 rbfs_make_partition.c File Reference

`#include "rbfs_make_partition.h"`
Include dependency graph for rbfs_make_partition.c:



Macros

- #define __RBFS_MAKE_PARTITION_C__ 1

Functions

- void rbfs_make_partition (rbfs_t *rbfs, rbfs_disk_size_t size, rbfs_environment_t env, rbfs_driver_t driver)

4.9.1 Macro Definition Documentation

4.9.1.1 #define __RBFS_MAKE_PARTITION_C__ 1

rbfs - Simple Resource Based File System

rbfs_init_partition.c

Initializes a rbfs partition

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file rbfs_make_partition.c.

4.9.2 Function Documentation

4.9.2.1 void rbfs_make_partition (rbfs_t * *rbfs*, rbfs_disk_size_t *size*, rbfs_environment_t *env*, rbfs_driver_t *driver*)

Definition at line 16 of file rbfs_make_partition.c.

4.10 rbfs_make_partition.c

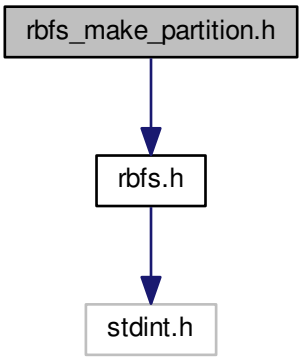
```
00001
00011 #ifndef __RBFS_MAKE_PARTITION_C__
00012 #define __RBFS_MAKE_PARTITION_C__ 1
00013
00014 #include "rbfs_make_partition.h"
00015
00016 void rbfs_make_partition(rbfs_t *rbfs, rbfs_disk_size_t size,
      rbfs_environment_t env, rbfs_driver_t driver) {
00017
00018     rbfs->driver = driver;
00019     switch(size) {
00020
00021         case RBFS_DISK_32K:
00022             rbfs->memory_size = 0x7f94; //32660;
00023             rbfs->resource_descriptor_table_address = 0x0020; //32;
00024             rbfs->cluster_table_address = 0x00a0; //160;
00025             rbfs->sizeof_resource_descriptor_table = 0x0080; //128;
00026             rbfs->sizeof_cluster_table = 0x7ef4; //32500;
00027             rbfs->sizeof_resource_descriptor = 0x04; //4;
00028             rbfs->sizeof_cluster = 0x82; //130;
00029             rbfs->resource_descriptor_count = 0x20; //32;
00030             rbfs->cluster_count = 0xfa; //250;
00031             rbfs->sizeof_cluster_data = 0x80; //128;
00032             rbfs->sizeof_cluster_control = 0x02; //2;
00033             rbfs->free_clusters = 0xfa; //250;
00034         break;
00035
00036         case RBFS_DISK_8K:
00037             rbfs->memory_size = 0x2000; //8192;
00038             rbfs->resource_descriptor_table_address = 0x0020; //32;
00039             rbfs->cluster_table_address = 0x00a0; //160;
00040             rbfs->sizeof_resource_descriptor_table = 0x0080; //128;
00041             rbfs->sizeof_cluster_table = 0x1f60; //8032;
00042             rbfs->sizeof_resource_descriptor = 0x04; //4;
00043             rbfs->sizeof_cluster = 0x20; //32;
00044             rbfs->resource_descriptor_count = 0x20; //32;
00045             rbfs->cluster_count = 0xfb; //251;
00046             rbfs->sizeof_cluster_data = 0x1e; //30;
00047             rbfs->sizeof_cluster_control = 0x02; //2;
00048             rbfs->free_clusters = 0xfb; //251;
00049         break;
00050
00051         default:
```

```
00052             rbfs->memory_size = 0xf46; //3910;
00053             rbfs->resource_descriptor_table_address = 0x0020; //32;
00054             rbfs->cluster_table_address = 0x00a0; //160;
00055             rbfs->sizeof_resource_descriptor_table = 0x0080; //128;
00056             rbfs->sizeof_cluster_table = 0xea6; //3750;
00057             rbfs->sizeof_resource_descriptor = 0x04; //4;
00058             rbfs->sizeof_cluster = 0x0f; //32;
00059             rbfs->resource_descriptor_count = 0x20; //32;
00060             rbfs->cluster_count = 0xfa; //250;
00061             rbfs->sizeof_cluster_data = 0x0d; //13;
00062             rbfs->sizeof_cluster_control = 0x02; //2;
00063             rbfs->free_clusters = 0xfa; //250;
00064         break;
00065     }
00066     rbfs->flags = 0x00; //0;
00067 }
00068
00069 #endif // __RBFS_MAKE_PARTITION_C__
```

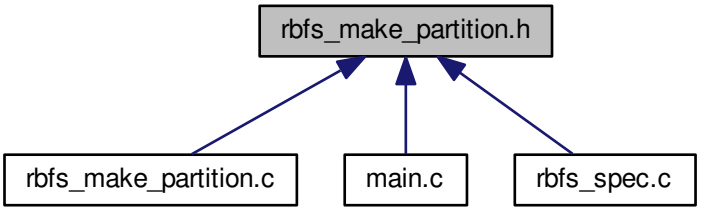
4.11 rbfs_make_partition.h File Reference

`#include "rbfs.h"`

Include dependency graph for rbfs_make_partition.h:



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



Enumerations

- enum rbfs_disk_size_t { RBFS_DISK_4K, RBFS_DISK_8K, RBFS_DISK_32K }
- enum rbfs_environment_t { RBFS_ENV_ARDUINO, RBFS_ENV_VIRTUAL }

Functions

- void rbfs_make_partition (rbfs_t *rbfs, rbfs_disk_size_t size, rbfs_environment_t env, rbfs_driver_t driver)

4.11.1 Enumeration Type Documentation

4.11.1.1 enum rbfs_disk_size_t

rbfs - Simple Resource Based File System

rbfs_init_partition.h

Initializes a rbfs partition

Author

Dalmir da Silva dalmirdasilva@gmail.com

Enumerator

RBFS_DISK_4K

RBFS_DISK_8K

RBFS_DISK_32K

Definition at line 16 of file rbfs_make_partition.h.

4.11.1.2 enum rbfs_environment_t

Enumerator

RBFS_ENV_ARDUINO

RBFS_ENV_VIRTUAL

Definition at line 22 of file rbfs_make_partition.h.

4.11.2 Function Documentation

4.11.2.1 void rbfs_make_partition (rbfs_t * *rbfs*, rbfs_disk_size_t *size*, rbfs_environment_t *env*, rbfs_driver_t *driver*)

Definition at line 16 of file rbfs_make_partition.c.

4.12 rbfs_make_partition.h

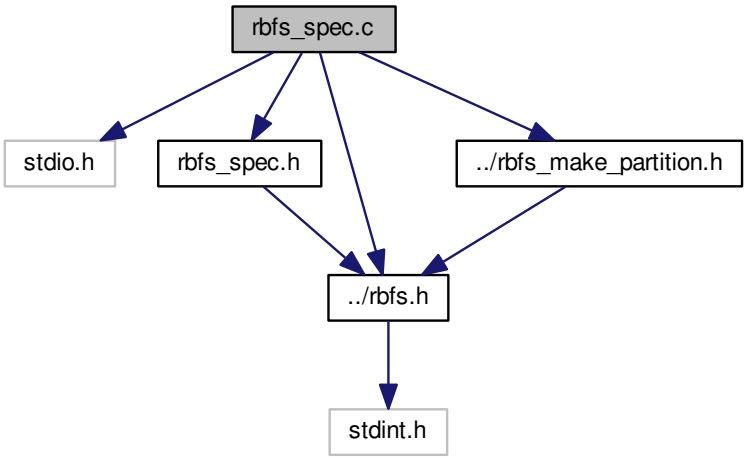
```
00001
00011 #ifndef __RBFS_MAKE_PARTITION_H__
00012 #define __RBFS_MAKE_PARTITION_H__ 1
00013
00014 #include "rbfs.h"
00015
00016 typedef enum {
00017     RBFS_DISK_4K,
00018     RBFS_DISK_8K,
00019     RBFS_DISK_32K
00020 } rbfs_disk_size_t;
00021
```

```
00022 typedef enum {
00023     RBFS_ENV_ARDUINO,
00024     RBFS_ENV_VIRTUAL
00025 } rbfs_environment_t;
00026
00027 void rbfs_make_partition(rbfs_t *rbfs, rbfs_disk_size_t size,
      rbfs_environment_t env, rbfs_driver_t driver);
00028
00029 #endif // __RBFS_MAKE_PARTITION_H__
```

4.13 rbfs_spec.c File Reference

```
#include <stdio.h>
#include "rbfs_spec.h"
#include "../rbfs.h"
#include "../rbfs_make_partition.h"
```

Include dependency graph for rbfs_spec.c:



Functions

- void format_spec (rbfs_t *rbfs)
- void mount_spec (rbfs_t *rbfs)
- void umount_spec (rbfs_t *rbfs)
- void alloc_resource_spec (rbfs_t *rbfs)
- void try_to_alloc_resources_that_is_possible_spec (rbfs_t *rbfs)
- void open_resource_spec (rbfs_t *rbfs)
- void write_resource_spec (rbfs_t *rbfs)
- void rewind_resource_spec (rbfs_t *rbfs)
- void read_resource_spec (rbfs_t *rbfs)
- void close_resource_spec (rbfs_t *rbfs)
- void try_read_when_end_of_resource_is_reached_spec (rbfs_t *rbfs)
- void try_read_when_resource_is_closed_spec (rbfs_t *rbfs)
- void seek_resource_spec (rbfs_t *rbfs)
- void random_read_resource_spec (rbfs_t *rbfs)
- void random_read_with_seek_resource_spec (rbfs_t *rbfs)
- void random_read_with_seek_opening_resource_spec (rbfs_t *rbfs)

- void size_resource_spec (rbfs_t *rbfs)
- void tell_resource_spec (rbfs_t *rbfs)
- void tell_with_seek_resource_spec (rbfs_t *rbfs)
- void total_space_resource_spec (rbfs_t *rbfs)
- void allocating_multi_format_spec (rbfs_t *rbfs)
- void read_only_mounting_spec (rbfs_t *rbfs)
- void read_only_opening_spec (rbfs_t *rbfs)

4.13.1 Function Documentation

4.13.1.1 void alloc_resource_spec (rbfs_t * *rbfs*)

Definition at line 43 of file rbfs_spec.c.

4.13.1.2 void allocating_multi_format_spec (rbfs_t * *rbfs*)

Definition at line 442 of file rbfs_spec.c.

4.13.1.3 void close_resource_spec (rbfs_t * *rbfs*)

Definition at line 154 of file rbfs_spec.c.

4.13.1.4 void format_spec (rbfs_t * *rbfs*)

Definition at line 6 of file rbfs_spec.c.

4.13.1.5 void mount_spec (rbfs_t * *rbfs*)

Definition at line 17 of file rbfs_spec.c.

4.13.1.6 void open_resource_spec (rbfs_t * *rbfs*)

Definition at line 77 of file rbfs_spec.c.

4.13.1.7 void random_read_resource_spec (rbfs_t * *rbfs*)

Definition at line 234 of file rbfs_spec.c.

4.13.1.8 void random_read_with_seek_opening_resource_spec (rbfs_t * *rbfs*)

Definition at line 303 of file rbfs_spec.c.

4.13.1.9 void random_read_with_seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 266 of file rbfs_spec.c.

4.13.1.10 void read_only_mounting_spec (rbfs_t * *rbfs*)

Definition at line 472 of file rbfs_spec.c.

4.13.1.11 void read_only_opening_spec (rbfs_t * *rbfs*)

Definition at line 497 of file rbfs_spec.c.

4.13.1.12 void read_resource_spec (rbfs_t * *rbfs*)

Definition at line 131 of file rbfs_spec.c.

4.13.1.13 void rewind_resource_spec (rbfs_t * *rbfs*)

Definition at line 112 of file rbfs_spec.c.

4.13.1.14 void seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 212 of file rbfs_spec.c.

4.13.1.15 void size_resource_spec (rbfs_t * *rbfs*)

Definition at line 343 of file rbfs_spec.c.

4.13.1.16 void tell_resource_spec (rbfs_t * *rbfs*)

Definition at line 365 of file rbfs_spec.c.

4.13.1.17 void tell_with_seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 386 of file rbfs_spec.c.

4.13.1.18 void total_space_resource_spec (rbfs_t * *rbfs*)

Definition at line 418 of file rbfs_spec.c.

4.13.1.19 void try_read_when_end_of_resource_is_reached_spec (rbfs_t * *rbfs*)

Definition at line 172 of file rbfs_spec.c.

4.13.1.20 void try_read_when_resource_is_closed_spec (rbfs_t * *rbfs*)

Definition at line 193 of file rbfs_spec.c.

4.13.1.21 void try_to_alloc_resources_that_is_possible_spec (rbfs_t * *rbfs*)

Definition at line 58 of file rbfs_spec.c.

4.13.1.22 void umount_spec (rbfs_t * *rbfs*)

Definition at line 30 of file rbfs_spec.c.

4.13.1.23 void write_resource_spec (rbfs_t * *rbfs*)

Definition at line 94 of file rbfs_spec.c.

4.14 rbfs_spec.c

```
00001 #include <stdio.h>
00002 #include "rbfs_spec.h"
00003 #include "../rbfs.h"
00004 #include "../rbfs_make_partition.h"
00005
00006 void format_spec(rbfs_t *rbfs) {
00007     rbfs_op_result_t op_r;
00008     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00009     op_r = rbfs_format(rbfs);
00010     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00011         RBFS_SPEC_PRINTF("(F) format spec failed. error: %d\n", op_r);
00012     } else {
00013         RBFS_SPEC_PRINTF("(*) format spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00014     }
00015 }
00016
00017 void mount_spec(rbfs_t *rbfs) {
00018     rbfs_op_result_t op_r;
00019     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00020     op_r = rbfs_format(rbfs);
00021     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00022     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00023         RBFS_SPEC_PRINTF("(F) mount spec failed. error: %d\n", op_r);
```

```
00024     } else {
00025         RBFS_SPEC_PRINTF("(*) mount spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00026     }
00027     rbfs_umount(rbfs);
00028 }
00029
00030 void umount_spec(rbfs_t *rbfs) {
00031     rbfs_op_result_t op_r;
00032     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00033     op_r = rbfs_format(rbfs);
00034     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00035     op_r = rbfs_umount(rbfs);
00036     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00037         RBFS_SPEC_PRINTF("(F) unmount spec failed. error: %d\n", op_r);
00038     } else {
00039         RBFS_SPEC_PRINTF("(*) unmount spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00040     }
00041 }
00042
00043 void alloc_resource_spec(rbfs_t *rbfs) {
00044     rbfs_op_result_t op_r;
00045     rbfs_resource_code_t rbfs_resource_code;
00046     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00047     op_r = rbfs_format(rbfs);
00048     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00049     rbfs_resource_code = rbfs_alloc(rbfs);
00050     if (rbfs_resource_code == RBFS_NULL_RESOURCE_CODE) {
00051         RBFS_SPEC_PRINTF("(F) alloc_resource spec failed. error: %d\n", op_r);
00052     } else {
00053         RBFS_SPEC_PRINTF("(*) alloc_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00054     }
00055     rbfs_umount(rbfs);
00056 }
00057
00058 void try_to_alloc_resources_that_is_possible_spec(
      rbfs_t *rbfs) {
00059     rbfs_op_result_t op_r;
00060     rbfs_resource_code_t rbfs_resource_code[2] = { 0x00, 0x00 };
00061     uint8_t i = 0;
00062     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00063     op_r = rbfs_format(rbfs);
00064     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00065     for (; i < rbfs->resource_descriptor_count; i++) {
00066         rbfs_resource_code[0] = rbfs_alloc(rbfs);
00067     }
00068     rbfs_resource_code[1] = rbfs_alloc(rbfs);
00069     if (rbfs_resource_code[0] == (rbfs->resource_descriptor_count - 1) &&
      rbfs_resource_code[1] == RBFS_NULL_RESOURCE_CODE) {
00070         RBFS_SPEC_PRINTF("(*) try_to_alloc_resources_that_is_possible spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00071     } else {
00072         RBFS_SPEC_PRINTF("(F) try_to_alloc_resources_that_is_possible spec failed. error:
       %x\n", rbfs_resource_code[0]);
00073     }
00074     rbfs_umount(rbfs);
00075 }
00076
00077 void open_resource_spec(rbfs_t *rbfs) {
00078     rbfs_op_result_t op_r;
00079     rbfs_resource_code_t rbfs_resource_code;
00080     rbfs_resource_t resource;
00081     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00082     op_r = rbfs_format(rbfs);
00083     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00084     rbfs_resource_code = rbfs_alloc(rbfs);
00085     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00086     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00087         RBFS_SPEC_PRINTF("(F) open_resource spec failed. error: %d\n", op_r);
00088     } else {
00089         RBFS_SPEC_PRINTF("(*) open_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00090     }
00091     rbfs_umount(rbfs);
00092 }
00093
```

```
00094 void write_resource_spec(rbfs_t *rbfs) {
00095     rbfs_op_result_t op_r;
00096     rbfs_resource_code_t rbfs_resource_code;
00097     rbfs_resource_t resource;
00098     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00099     op_r = rbfs_format(rbfs);
00100     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00101     rbfs_resource_code = rbfs_alloc(rbfs);
00102     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00103     op_r = rbfs_write(rbfs, &resource, 0xaa);
00104     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00105         RBFS_SPEC_PRINTF("(F) write_resource spec failed. error: %d\n", op_r);
00106     } else {
00107         RBFS_SPEC_PRINTF("(*) write_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00108     }
00109     rbfs_umount(rbfs);
00110 }
00111
00112 void rewind_resource_spec(rbfs_t *rbfs) {
00113     rbfs_op_result_t op_r;
00114     rbfs_resource_code_t rbfs_resource_code;
00115     rbfs_resource_t resource;
00116     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00117     op_r = rbfs_format(rbfs);
00118     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00119     rbfs_resource_code = rbfs_alloc(rbfs);
00120     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00121     op_r = rbfs_write(rbfs, &resource, 0xAA);
00122     op_r = rbfs_rewind(rbfs, &resource);
00123     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00124         RBFS_SPEC_PRINTF("(F) rewind_resource spec failed. error: %d\n", op_r);
00125     } else {
00126         RBFS_SPEC_PRINTF("(*) rewind_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00127     }
00128     rbfs_umount(rbfs);
00129 }
00130
00131 void read_resource_spec(rbfs_t *rbfs) {
00132     rbfs_op_result_t op_r;
00133     rbfs_resource_code_t rbfs_resource_code;
00134     rbfs_resource_t resource;
00135     unsigned char c[2];
00136     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00137     op_r = rbfs_format(rbfs);
00138     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00139     rbfs_resource_code = rbfs_alloc(rbfs);
00140     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00141     op_r = rbfs_write(rbfs, &resource, 0x41);
00142     op_r = rbfs_write(rbfs, &resource, 0xA1);
00143     op_r = rbfs_rewind(rbfs, &resource);
00144     c[0] = rbfs_read(rbfs, &resource);
00145     c[1] = rbfs_read(rbfs, &resource);
00146     if (c[0] != 0x41 || c[1] != 0xA1) {
00147         RBFS_SPEC_PRINTF("(F) read_resource spec failed. error: %x\n", c[0]);
00148     } else {
00149         RBFS_SPEC_PRINTF("(*) read_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00150     }
00151     rbfs_umount(rbfs);
00152 }
00153
00154 void close_resource_spec(rbfs_t *rbfs) {
00155     rbfs_op_result_t op_r;
00156     rbfs_resource_code_t rbfs_resource_code;
00157     rbfs_resource_t resource;
00158     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00159     op_r = rbfs_format(rbfs);
00160     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00161     rbfs_resource_code = rbfs_alloc(rbfs);
00162     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00163     op_r = rbfs_close(rbfs, &resource);
00164     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00165         RBFS_SPEC_PRINTF("(F) close_resource spec failed. error: %x\n", op_r);
```

```
00166     } else {
00167         RBFS_SPEC_PRINTF("(*) close_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00168     }
00169     rbfs_umount(rbfs);
00170 }
00171
00172 void try_read_when_end_of_resource_is_reached_spec(
      rbfs_t *rbfs) {
00173     rbfs_op_result_t op_r;
00174     rbfs_resource_code_t rbfs_resource_code;
00175     rbfs_resource_t resource;
00176     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00177     op_r = rbfs_format(rbfs);
00178     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00179     rbfs_resource_code = rbfs_alloc(rbfs);
00180     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00181     op_r = rbfs_write(rbfs, &resource, 0x41);
00182     op_r = rbfs_rewind(rbfs, &resource);
00183     rbfs_read(rbfs, &resource);
00184     rbfs_read(rbfs, &resource);
00185     if (op_r == 0 && (rbfs_eor(&resource))) {
00186         RBFS_SPEC_PRINTF("(*) try_read_when_end_of_resource_is_reached spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00187     } else {
00188         RBFS_SPEC_PRINTF("(F) try_read_when_end_of_resource_is_reached spec failed. error:
       %x\n", op_r);
00189     }
00190     rbfs_umount(rbfs);
00191 }
00192
00193 void try_read_when_resource_is_closed_spec(
      rbfs_t *rbfs) {
00194     rbfs_op_result_t op_r;
00195     rbfs_resource_code_t rbfs_resource_code;
00196     rbfs_resource_t resource;
00197     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00198     op_r = rbfs_format(rbfs);
00199     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00200     rbfs_resource_code = rbfs_alloc(rbfs);
00201     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00202     op_r = rbfs_close(rbfs, &resource);
00203     rbfs_read(rbfs, &resource);
00204     if (op_r == 0 && (resource.flags |
      RBFS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ)) {
00205         RBFS_SPEC_PRINTF("(*) try_read_when_resource_is_closed spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00206     } else {
00207         RBFS_SPEC_PRINTF("(F) try_read_when_resource_is_closed spec failed. error: %x\n",
      op_r);
00208     }
00209     rbfs_umount(rbfs);
00210 }
00211
00212 void seek_resource_spec(rbfs_t *rbfs) {
00213     rbfs_op_result_t op_r;
00214     rbfs_resource_code_t rbfs_resource_code;
00215     rbfs_resource_t resource;
00216     uint8_t i = 0;
00217     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00218     op_r = rbfs_format(rbfs);
00219     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00220     rbfs_resource_code = rbfs_alloc(rbfs);
00221     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00222     for (; i < 50; i++) {
00223         op_r = rbfs_write(rbfs, &resource, (i + 0x65));
00224     }
00225     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 20);
00226     if (op_r == RBFS_OP_RESULT_SUCCESS) {
00227         RBFS_SPEC_PRINTF("(*) seek_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00228     } else {
00229         RBFS_SPEC_PRINTF("(F) seek_resource spec failed. error: %x\n", op_r);
00230     }
00231     rbfs_umount(rbfs);
00232 }
00233
00234 void random_read_resource_spec(rbfs_t *rbfs) {
```

```
00235     rbfs_op_result_t op_r;
00236     rbfs_resource_code_t rbfs_resource_code;
00237     rbfs_resource_t resource;
00238     uint8_t i = 0;
00239     unsigned char c[5], first_write_char = 0x65;
00240     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00241     op_r = rbfs_format(rbfs);
00242     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00243     rbfs_resource_code = rbfs_alloc(rbfs);
00244     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00245     for (; i < 255; i++) {
00246         op_r = rbfs_write(rbfs, &resource, (i + first_write_char));
00247     }
00248     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 0x20);
00249     c[0] = rbfs_read(rbfs, &resource);
00250     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_CURRENT, 0x10);
00251     c[1] = rbfs_read(rbfs, &resource);
00252     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 0x48);
00253     c[2] = rbfs_read(rbfs, &resource);
00254     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_CURRENT, 0x20);
00255     c[3] = rbfs_read(rbfs, &resource);
00256     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 0x0);
00257     c[4] = rbfs_read(rbfs, &resource);
00258     if (c[0] == first_write_char + 0x20 && c[1] == first_write_char + 0x31 && c[2] == first_write_char + 0
      x48 && c[3] == first_write_char + 0x69 && c[4] == first_write_char + 0x0) {
00259         RBFS_SPEC_PRINTF("(*) random_read_resource spec passed, RBFS_OP_RESULT_SUCCESS: %d.
      \n", RBFS_OP_RESULT_SUCCESS);
00260     } else {
00261         RBFS_SPEC_PRINTF("(F) random_read_resource spec failed. error: %x\n", op_r);
00262     }
00263     rbfs_umount(rbfs);
00264 }
00265
00266 void random_read_with_seek_resource_spec(
      rbfs_t *rbfs) {
00267     rbfs_op_result_t op_r;
00268     rbfs_resource_code_t rbfs_resource_code;
00269     rbfs_resource_t resource;
00270     uint8_t i = 0;
00271     unsigned char c[255];
00272     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00273     op_r = rbfs_format(rbfs);
00274     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00275     rbfs_resource_code = rbfs_alloc(rbfs);
00276     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00277     for (i = 0; i < 255; i++) {
00278         op_r = rbfs_write(rbfs, &resource, i);
00279     }
00280     rbfs_rewind(rbfs, &resource);
00281     for (i = 0; i < 255; i++) {
00282         rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, i);
00283         c[i] = rbfs_read(rbfs, &resource);
00284         rbfs_read(rbfs, &resource);
00285         rbfs_read(rbfs, &resource);
00286         rbfs_read(rbfs, &resource);
00287     }
00288     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 199);
00289     if ((i = rbfs_read(rbfs, &resource)) != 199) {
00290         RBFS_SPEC_PRINTF("(F) random_read_with_seek_resource_spec spec failed. %d != 199\n"
      , 0);
00291     }
00292
00293     for (i = 0; i < 255; i++) {
00294         if (i != c[i]) {
00295             RBFS_SPEC_PRINTF("(F) random_read_with_seek_resource_spec spec failed. error:
       %x\n", i);
00296         }
00297     }
00298
00299     RBFS_SPEC_PRINTF("(*) random_read_with_seek_resource_spec spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00300     rbfs_umount(rbfs);
00301 }
00302
00303 void random_read_with_seek_opening_resource_spec(
      rbfs_t *rbfs) {
00304     rbfs_op_result_t op_r;
00305     rbfs_resource_code_t rbfs_resource_code;
00306     rbfs_resource_t resource;
00307     uint8_t i = 0;
00308     unsigned char c[255];
```

```
00309     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00310     op_r = rbfs_format(rbfs);
00311     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00312     rbfs_resource_code = rbfs_alloc(rbfs);
00313     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00314     for (i = 0; i < 255; i++) {
00315         op_r = rbfs_write(rbfs, &resource, i);
00316     }
00317     rbfs_close(rbfs, &resource);
00318
00319     for (i = 0; i < 255; i++) {
00320         op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00321         rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, i);
00322         c[i] = rbfs_read(rbfs, &resource);
00323         rbfs_read(rbfs, &resource);
00324         rbfs_read(rbfs, &resource);
00325         rbfs_read(rbfs, &resource);
00326         rbfs_close(rbfs, &resource);
00327     }
00328     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00329     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 199);
00330     if ((i = rbfs_read(rbfs, &resource)) != 199) {
00331         RBFS_SPEC_PRINTF("(F) random_read_with_seek_opening_resource_spec spec failed. %d
       != 199\n", op_r);
00332     }
00333     rbfs_close(rbfs, &resource);
00334     for (i = 0; i < 255; i++) {
00335         if (i != c[i]) {
00336             RBFS_SPEC_PRINTF("(F) random_read_with_seek_opening_resource_spec spec failed.
       error: %x\n", c[i]);
00337         }
00338     }
00339     RBFS_SPEC_PRINTF("(*) random_read_with_seek_opening_resource_spec spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00340     rbfs_umount(rbfs);
00341 }
00342
00343 void size_resource_spec(rbfs_t *rbfs) {
00344     rbfs_op_result_t op_r;
00345     rbfs_resource_code_t rbfs_resource_code;
00346     rbfs_resource_t resource;
00347     uint16_t i = 0;
00348     uint16_t size = 0xf40;
00349     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00350     op_r = rbfs_format(rbfs);
00351     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00352     rbfs_resource_code = rbfs_alloc(rbfs);
00353     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00354     for (; i < size; i++) {
00355         op_r = rbfs_write(rbfs, &resource, 0x65);
00356     }
00357     if (rbfs_size(&resource) == 0xf40) {
00358         RBFS_SPEC_PRINTF("(*) size_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00359     } else {
00360         RBFS_SPEC_PRINTF("(F) size_resource spec failed. error: %x\n", size);
00361     }
00362     rbfs_umount(rbfs);
00363 }
00364
00365 void tell_resource_spec(rbfs_t *rbfs) {
00366     rbfs_op_result_t op_r;
00367     rbfs_resource_code_t rbfs_resource_code;
00368     rbfs_resource_t resource;
00369     uint8_t i = 0;
00370     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00371     op_r = rbfs_format(rbfs);
00372     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00373     rbfs_resource_code = rbfs_alloc(rbfs);
00374     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00375     for (; i < 50; i++) {
00376         op_r = rbfs_write(rbfs, &resource, 0x65);
00377     }
00378     if (rbfs_tell(&resource) == 50) {
00379         RBFS_SPEC_PRINTF("(*) tell_resource spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
```

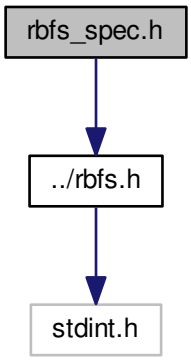
```
00380     } else {
00381         RBFS_SPEC_PRINTF("(F) tell_resource spec failed. error: %x\n", op_r);
00382     }
00383     rbfs_umount(rbfs);
00384 }
00385
00386 void tell_with_seek_resource_spec(rbfs_t *rbfs) {
00387     rbfs_op_result_t op_r;
00388     rbfs_resource_code_t rbfs_resource_code;
00389     rbfs_resource_t resource;
00390     uint8_t i = 0;
00391     rbfs_resource_size_t s[5];
00392     rbfs_make_partition(rbfs, RBFS_DISK_32K,
    RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00393     op_r = rbfs_format(rbfs);
00394     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
    RBFS_MOUNT_OPTION_NORMAL);
00395     rbfs_resource_code = rbfs_alloc(rbfs);
00396     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
    RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00397     for (; i < 50; i++) {
00398         op_r = rbfs_write(rbfs, &resource, 0x65);
00399     }
00400     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 20);
00401     s[0] = rbfs_tell(&resource);
00402     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_CURRENT, 10);
00403     s[1] = rbfs_tell(&resource);
00404     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 48);
00405     s[2] = rbfs_tell(&resource);
00406     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_CURRENT, 20);
00407     s[3] = rbfs_tell(&resource);
00408     op_r = rbfs_seek(rbfs, &resource, RBFS_SEEK_ORIGIN_BEGIN, 0);
00409     s[4] = rbfs_tell(&resource);
00410     if (s[0] == 20 && s[1] == 30 && s[2] == 48 && s[3] == 17 && s[4] == 0) {
00411         RBFS_SPEC_PRINTF("(*) tell_with_seek_resource spec passed %d.\n",
    RBFS_OP_RESULT_SUCCESS);
00412     } else {
00413         RBFS_SPEC_PRINTF("(F) tell_with_seek_resource spec failed. error: %d\n", s[3]);
00414     }
00415     rbfs_umount(rbfs);
00416 }
00417
00418 void total_space_resource_spec(rbfs_t *rbfs) {
00419     rbfs_op_result_t op_r;
00420     rbfs_resource_code_t rbfs_resource_code;
00421     rbfs_resource_t resource;
00422     rbfs_resource_size_t total_space[2];
00423     uint16_t i = 0;
00424     rbfs_make_partition(rbfs, RBFS_DISK_32K,
    RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00425     op_r = rbfs_format(rbfs);
00426     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
    RBFS_MOUNT_OPTION_NORMAL);
00427     total_space[0] = rbfs_available_space(rbfs);
00428     rbfs_resource_code = rbfs_alloc(rbfs);
00429     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
    RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00430     for (; i < rbfs->sizeof_cluster_data + 1; i++) {
00431         op_r = rbfs_write(rbfs, &resource, 0x65);
00432     }
00433     total_space[1] = rbfs_available_space(rbfs);
00434     if (total_space[0] - total_space[1] == (rbfs->sizeof_cluster_data * 2)) {
00435         RBFS_SPEC_PRINTF("(*) total_space_resource spec passed %d.\n",
    RBFS_OP_RESULT_SUCCESS);
00436     } else {
00437         RBFS_SPEC_PRINTF("(F) total_space_resource spec failed. error: %d != 50\n",
    total_space[0] - total_space[1]);
00438     }
00439     rbfs_umount(rbfs);
00440 }
00441
00442 void allocating_multi_format_spec(rbfs_t *rbfs) {
00443     rbfs_op_result_t op_r;
00444     rbfs_resource_t resource;
00445     uint8_t count = 3;
00446     uint8_t j, i;
00447     rbfs_resource_code_t rbfs_resource_code[3];
00448     uint8_t passed = 1;
00449     for (j = 0; j < count; j++) {
00450         rbfs_make_partition(rbfs, RBFS_DISK_32K,
    RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00451         op_r = rbfs_format(rbfs);
00452         op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
    RBFS_MOUNT_OPTION_NORMAL);
00453         rbfs_resource_code[j] = rbfs_alloc(rbfs);
00454         op_r = rbfs_open(rbfs, rbfs_resource_code[j], &resource,
    RBFS_OPEN_RESOURCE_OPTION_NORMAL);
```

```
00455         for (i = 0; i < 50; i++) {
00456             op_r = rbfs_write(rbfs, &resource, 0x65);
00457         }
00458         rbfs_close(rbfs, &resource);
00459     }
00460     for (j = 0; j < count; j++) {
00461         if (rbfs_resource_code[j] != 0) {
00462             RBFS_SPEC_PRINTF("(F) allocating_multi_format spec failed %x\n",
      rbfs_resource_code[j]);
00463             passed = 0;
00464         }
00465     }
00466     if (passed) {
00467         RBFS_SPEC_PRINTF("(*) allocating_multi_format spec passed %d\n",
      RBFS_OP_RESULT_SUCCESS);
00468     }
00469     rbfs_umount(rbfs);
00470 }
00471
00472 void read_only_mounting_spec(rbfs_t *rbfs) {
00473     rbfs_op_result_t op_r;
00474     rbfs_resource_t resource;
00475     rbfs_resource_code_t rbfs_resource_code;
00476     char error_msg[] = "(F) read_only_mounting spec failed. %d\n";
00477     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00478     op_r = rbfs_format(rbfs);
00479     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_READ_ONLY);
00480     rbfs_resource_code = rbfs_alloc(rbfs);
00481     if (rbfs_resource_code == RBFS_NULL_RESOURCE_CODE) {
00482         RBFS_SPEC_PRINTF(error_msg, op_r);
00483     }
00484     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_NORMAL);
00485     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00486         RBFS_SPEC_PRINTF(error_msg, op_r);
00487     }
00488     op_r = rbfs_write(rbfs, &resource, 0xaa);
00489     if (op_r == RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
00490         RBFS_SPEC_PRINTF("(*) read_only_mounting spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00491     } else {
00492         RBFS_SPEC_PRINTF(error_msg, op_r);
00493     }
00494     rbfs_umount(rbfs);
00495 }
00496
00497 void read_only_opening_spec(rbfs_t *rbfs) {
00498     rbfs_op_result_t op_r;
00499     rbfs_resource_t resource;
00500     rbfs_resource_code_t rbfs_resource_code;
00501     char error_msg[] = "(F) read_only_opening spec failed. %d\n";
00502     rbfs_make_partition(rbfs, RBFS_DISK_32K,
      RBFS_ENV_VIRTUAL, RBFS_DRIVER_VIRTUAL);
00503     op_r = rbfs_format(rbfs);
00504     op_r = rbfs_mount(RBFS_DRIVER_VIRTUAL, rbfs,
      RBFS_MOUNT_OPTION_NORMAL);
00505     rbfs_resource_code = rbfs_alloc(rbfs);
00506     if (rbfs_resource_code == RBFS_NULL_RESOURCE_CODE) {
00507         RBFS_SPEC_PRINTF(error_msg, op_r);
00508     }
00509     op_r = rbfs_open(rbfs, rbfs_resource_code, &resource,
      RBFS_OPEN_RESOURCE_OPTION_READ_ONLY);
00510     if (op_r != RBFS_OP_RESULT_SUCCESS) {
00511         RBFS_SPEC_PRINTF(error_msg, op_r);
00512     }
00513     op_r = rbfs_write(rbfs, &resource, 0xaa);
00514     if (op_r == RBFS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
00515         RBFS_SPEC_PRINTF("(*) read_only_opening spec passed %d.\n",
      RBFS_OP_RESULT_SUCCESS);
00516     } else {
00517         RBFS_SPEC_PRINTF(error_msg, op_r);
00518     }
00519     rbfs_umount(rbfs);
00520 }
```

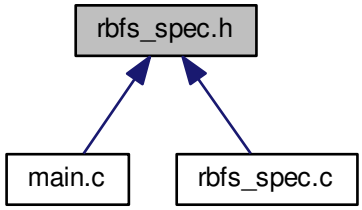
4.15 rbfs_spec.h File Reference

```
#include "../rbfs.h"
```

Include dependency graph for rbfs_spec.h:



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



Macros

- #define RBFS_SPEC_PRINTF printf

Functions

- void format_spec (rbfs_t *rbfs)
- void mount_spec (rbfs_t *rbfs)
- void umount_spec (rbfs_t *rbfs)
- void alloc_resource_spec (rbfs_t *rbfs)
- void try_to_alloc_resources_that_is_possible_spec (rbfs_t *rbfs)
- void open_resource_spec (rbfs_t *rbfs)
- void write_resource_spec (rbfs_t *rbfs)
- void rewind_resource_spec (rbfs_t *rbfs)

- void read_resource_spec (rbfs_t *rbfs)
- void close_resource_spec (rbfs_t *rbfs)
- void try_read_when_end_of_resource_is_reached_spec (rbfs_t *rbfs)
- void try_read_when_resource_is_closed_spec (rbfs_t *rbfs)
- void seek_resource_spec (rbfs_t *rbfs)
- void random_read_resource_spec (rbfs_t *rbfs)
- void random_read_with_seek_resource_spec (rbfs_t *rbfs)
- void random_read_with_seek_opening_resource_spec (rbfs_t *rbfs)
- void size_resource_spec (rbfs_t *rbfs)
- void tell_resource_spec (rbfs_t *rbfs)
- void tell_with_seek_resource_spec (rbfs_t *rbfs)
- void total_space_resource_spec (rbfs_t *rbfs)
- void allocating_multi_format_spec (rbfs_t *rbfs)
- void read_only_mounting_spec (rbfs_t *rbfs)
- void read_only_opening_spec (rbfs_t *rbfs)

4.15.1 Macro Definition Documentation

4.15.1.1 #define RBFS_SPEC_PRINTF printf

Definition at line 4 of file rbfs_spec.h.

4.15.2 Function Documentation

4.15.2.1 void alloc_resource_spec (rbfs_t * *rbfs*)

Definition at line 43 of file rbfs_spec.c.

4.15.2.2 void allocating_multi_format_spec (rbfs_t * *rbfs*)

Definition at line 442 of file rbfs_spec.c.

4.15.2.3 void close_resource_spec (rbfs_t * *rbfs*)

Definition at line 154 of file rbfs_spec.c.

4.15.2.4 void format_spec (rbfs_t * *rbfs*)

Definition at line 6 of file rbfs_spec.c.

4.15.2.5 void mount_spec (rbfs_t * *rbfs*)

Definition at line 17 of file rbfs_spec.c.

4.15.2.6 void open_resource_spec (rbfs_t * *rbfs*)

Definition at line 77 of file rbfs_spec.c.

4.15.2.7 void random_read_resource_spec (rbfs_t * *rbfs*)

Definition at line 234 of file rbfs_spec.c.

4.15.2.8 void random_read_with_seek_opening_resource_spec (rbfs_t * *rbfs*)

Definition at line 303 of file rbfs_spec.c.

4.15.2.9 void random_read_with_seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 266 of file rbfs_spec.c.

4.15.2.10 void read_only_mounting_spec (rbfs_t * *rbfs*)

Definition at line 472 of file rbfs_spec.c.

4.15.2.11 void read_only_opening_spec (rbfs_t * *rbfs*)

Definition at line 497 of file rbfs_spec.c.

4.15.2.12 void read_resource_spec (rbfs_t * *rbfs*)

Definition at line 131 of file rbfs_spec.c.

4.15.2.13 void rewind_resource_spec (rbfs_t * *rbfs*)

Definition at line 112 of file rbfs_spec.c.

4.15.2.14 void seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 212 of file rbfs_spec.c.

4.15.2.15 void size_resource_spec (rbfs_t * *rbfs*)

Definition at line 343 of file rbfs_spec.c.

4.15.2.16 void tell_resource_spec (rbfs_t * *rbfs*)

Definition at line 365 of file rbfs_spec.c.

4.15.2.17 void tell_with_seek_resource_spec (rbfs_t * *rbfs*)

Definition at line 386 of file rbfs_spec.c.

4.15.2.18 void total_space_resource_spec (rbfs_t * *rbfs*)

Definition at line 418 of file rbfs_spec.c.

4.15.2.19 void try_read_when_end_of_resource_is_reached_spec (rbfs_t * *rbfs*)

Definition at line 172 of file rbfs_spec.c.

4.15.2.20 void try_read_when_resource_is_closed_spec (rbfs_t * *rbfs*)

Definition at line 193 of file rbfs_spec.c.

4.15.2.21 void try_to_alloc_resources_that_is_possible_spec (rbfs_t * *rbfs*)

Definition at line 58 of file rbfs_spec.c.

4.15.2.22 void umount_spec (rbfs_t * *rbfs*)

Definition at line 30 of file rbfs_spec.c.

4.15.2.23 void write_resource_spec (rbfs_t * *rbfs*)

Definition at line 94 of file rbfs_spec.c.

4.16 rbfs_spec.h

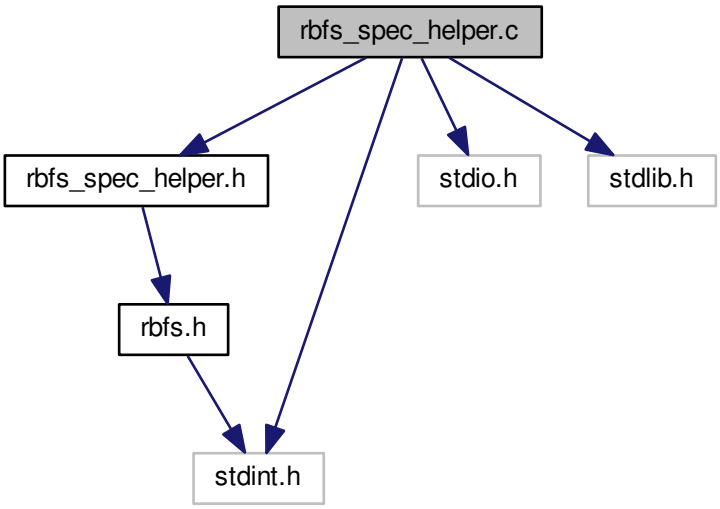
```
00001 #ifndef __RBFS_SPEC_H__
00002 #define __RBFS_SPEC_H__ 1
00003
00004 #define RBFS_SPEC_PRINTF printf
00005
```

```
00006 #include "../rbfs.h"
00007
00008 void format_spec(rbfs_t *rbfs);
00009
00010 void mount_spec(rbfs_t *rbfs);
00011
00012 void umount_spec(rbfs_t *rbfs);
00013
00014 void alloc_resource_spec(rbfs_t *rbfs);
00015
00016 void try_to_alloc_resources_that_is_possible_spec(
      rbfs_t *rbfs);
00017
00018 void open_resource_spec(rbfs_t *rbfs);
00019
00020 void write_resource_spec(rbfs_t *rbfs);
00021
00022 void rewind_resource_spec(rbfs_t *rbfs);
00023
00024 void read_resource_spec(rbfs_t *rbfs);
00025
00026 void close_resource_spec(rbfs_t *rbfs);
00027
00028 void try_read_when_end_of_resource_is_reached_spec(
      rbfs_t *rbfs);
00029
00030 void try_read_when_resource_is_closed_spec(
      rbfs_t *rbfs);
00031
00032 void seek_resource_spec(rbfs_t *rbfs);
00033
00034 void random_read_resource_spec(rbfs_t *rbfs);
00035
00036 void random_read_with_seek_resource_spec(
      rbfs_t *rbfs);
00037
00038 void random_read_with_seek_opening_resource_spec(
      rbfs_t *rbfs);
00039
00040 void size_resource_spec(rbfs_t *rbfs);
00041
00042 void tell_resource_spec(rbfs_t *rbfs);
00043
00044 void tell_with_seek_resource_spec(rbfs_t *rbfs);
00045
00046 void total_space_resource_spec(rbfs_t *rbfs);
00047
00048 void allocating_multi_format_spec(rbfs_t *rbfs);
00049
00050 void read_only_mounting_spec(rbfs_t *rbfs);
00051
00052 void read_only_opening_spec(rbfs_t *rbfs);
00053
00054 #endif // __RBFS_SPEC_H__
```

4.17 rbfs_spec_helper.c File Reference

```
#include "rbfs_spec_helper.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
```

Include dependency graph for rbfs_spec_helper.c:



Functions

- void resource_dump (rbfs_resource_t *resource)
- void format_all ()
- char * itob (int i)
- void rbfs_io_memory_dump (rbfs_t *rbfs)

4.17.1 Function Documentation

4.17.1.1 void format_all ( )

Definition at line 20 of file rbfs_spec_helper.c.

4.17.1.2 char* itob (int *i*)

Definition at line 27 of file rbfs_spec_helper.c.

4.17.1.3 void rbfs_io_memory_dump (rbfs_t * *rbfs*)

Definition at line 41 of file rbfs_spec_helper.c.

4.17.1.4 void resource_dump (rbfs_resource_t * *resource*)

Definition at line 7 of file rbfs_spec_helper.c.

4.18 rbfs_spec_helper.c

```
00001 #include "rbfs_spec_helper.h"
00002
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <stdint.h>
```

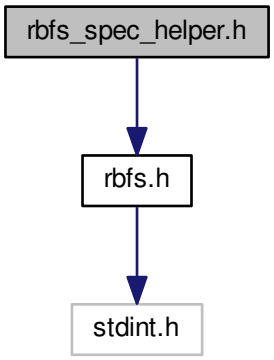
```
00006
00007 void resource_dump(rbfs_resource_t *resource) {
00008     printf("======== resource dump begin ========\n");
00009     printf("  resource descriptor: %6d %s\n", resource->resource_descriptor,
      itob(resource->resource_descriptor));
00010     printf("  first cluster:______ %6d %s\n", resource->first_cluster,
      itob(resource->first_cluster));
00011     printf("  current cluster:____ %6d %s\n", resource->current_cluster,
      itob(resource->current_cluster));
00012     printf("  cluster offset:_____ %6d %s\n", resource->cluster_offset,
      itob(resource->cluster_offset));
00013     printf("  size:_______________ %6d %s\n", resource->size, itob(resource->
      size));
00014     printf("  current position:___ %6d %s\n", resource->current_position,
      itob(resource->current_position));
00015     printf("  flags:______________ %6d %s\n", resource->flags, itob(resource->
      flags));
00016     printf("  errors:_____________ %6d %s\n", rbfs_error(resource),
      itob(rbfs_error(resource)));
00017     printf("========= resource dump end =========\n");
00018 }
00019
00020 void format_all() {
00021     uint16_t i;
00022     for (i = 0; i < 0x8000; i++) {
00023         _rbfs_io_write(RBFS_DRIVER_VIRTUAL, i, 0x00);
00024     }
00025 }
00026
00027 char* itob(int i) {
00028     int bits;
00029     int j, k;
00030     uint16_t mi = 0;
00031     mi |= i;
00032     static char buff[sizeof(mi) * 8 + 1];
00033     bits = sizeof(mi) * 8;
00034     for (j = bits - 1, k = 0; j >= 0; j--, k++) {
00035         buff[k] = ((mi >> j) & 0x01) + '0';
00036     }
00037     buff[bits] = '\0';
00038     return buff;
00039 }
00040
00041 void rbfs_io_memory_dump(rbfs_t *rbfs) {
00042     rbfs_memory_address_t memory_address;
00043     uint16_t count, count2;
00044     uint8_t data = 0;
00045     FILE *fp;
00046     if (!_rbfs_is_driver_monted(rbfs->driver)) {
00047         printf("rbfs not mounted yet\n");
00048         return;
00049     }
00050     fp = fopen("dump", "w+");
00051     fprintf(fp, "DRIVER: %x\n", rbfs->driver);
00052     fprintf(fp, "\n==========================\n");
00053     fprintf(fp, "\nrbfs\n");
00054     fprintf(fp, "----------------\n");
00055     fprintf(fp, "memory_size:                          0x%04x %4d %s\n", rbfs->
      memory_size, rbfs->memory_size, itob(rbfs->memory_size));
00056     fprintf(fp, "resource_descriptor_table_address: 0x%04x %4d %s\n", rbfs->
      resource_descriptor_table_address, rbfs->
      resource_descriptor_table_address, itob(rbfs->
      resource_descriptor_table_address));
00057     fprintf(fp, "cluster_table_address:             0x%04x %4d %s\n", rbfs->
      cluster_table_address, rbfs->cluster_table_address,
      itob(rbfs->cluster_table_address));
00058     fprintf(fp, "sizeof_resource_descriptor_table:  0x%04x %4d %s\n", rbfs->
      sizeof_resource_descriptor_table, rbfs->
      sizeof_resource_descriptor_table, itob(rbfs->
      sizeof_resource_descriptor_table));
00059     fprintf(fp, "sizeof_cluster_table:              0x%04x %4d %s\n", rbfs->
      sizeof_cluster_table, rbfs->sizeof_cluster_table,
      itob(rbfs->sizeof_resource_descriptor_table));
00060     fprintf(fp, "sizeof_resource_descriptor:        0x%04x %4d %s\n", rbfs->
      sizeof_resource_descriptor, rbfs->
      sizeof_resource_descriptor, itob(rbfs->
      sizeof_resource_descriptor));
00061     fprintf(fp, "sizeof_cluster:                    0x%04x %4d %s\n", rbfs->
      sizeof_cluster, rbfs->sizeof_cluster, itob(rbfs->
      sizeof_cluster));
00062     fprintf(fp, "resource_descriptor_count:         0x%04x %4d %s\n", rbfs->
      resource_descriptor_count, rbfs->
      resource_descriptor_count, itob(rbfs->
      resource_descriptor_count));
00063     fprintf(fp, "cluster_count:                     0x%04x %4d %s\n", rbfs->
      cluster_count, rbfs->cluster_count, itob(rbfs->
      cluster_count));
```

```
00064     fprintf(fp, "sizeof_cluster_data:               0x%04x %4d %s\n", rbfs->
      sizeof_cluster_data, rbfs->sizeof_cluster_data,
      itob(rbfs->sizeof_cluster_data));
00065     fprintf(fp, "sizeof_cluster_control:            0x%04x %4d %s\n", rbfs->
      sizeof_cluster_control, rbfs->sizeof_cluster_control,
      itob(rbfs->sizeof_cluster_control));
00066     fprintf(fp, "free_clusters:                     0x%04x %4d %s\n", rbfs->
      free_clusters, rbfs->free_clusters, itob(rbfs->
      free_clusters));
00067     fprintf(fp, "flags:                             0x%04x %4d %s\n", rbfs->
      flags, rbfs->flags, itob(rbfs->flags));
00068     fprintf(fp, "\n=========================\n");
00069     fprintf(fp, "\nResource table\n");
00070     fprintf(fp, "----------------\n");
00071     count = 0;
00072     for (memory_address = rbfs->resource_descriptor_table_address;
      memory_address < (rbfs->resource_descriptor_table_address + rbfs->
      sizeof_resource_descriptor_table); memory_address++) {
00073         if ((count % rbfs->sizeof_resource_descriptor) == 0) {
00074             fprintf(fp, "\n%02x: ", (count) ? count / rbfs->
      sizeof_resource_descriptor : 0);
00075         }
00076         fprintf(fp, "%02x ", _rbfs_io_read(rbfs->driver, memory_address));
00077         count++;
00078     }
00079     fprintf(fp, "\n=========================\n");
00080     fprintf(fp, "\nCluster table\n");
00081     fprintf(fp, "----------------\n");
00082     fprintf(fp, "\n    |nn |pp |");
00083     for (count = 0; count < rbfs->sizeof_cluster_data; count++) {
00084         fprintf(fp, "dd ");
00085     }
00086     fprintf(fp, "\n    ---------");
00087     for (count = 0; count < rbfs->sizeof_cluster_data; count++) {
00088         fprintf(fp, "---");
00089     }
00090     count = 0;
00091
00092     for (memory_address = rbfs->cluster_table_address; memory_address < (rbfs->
      cluster_table_address + rbfs->sizeof_cluster_table);
      memory_address++) {
00093         if ((count % rbfs->sizeof_cluster) == 0) {
00094             fprintf(fp, "\n%02x: |", (count) ? count / rbfs->sizeof_cluster : 0);
00095             count2 = 0;
00096         }
00097         if (count2 == 1 || count2 == 2) {
00098             fprintf(fp, "|");
00099         }
00100         fprintf(fp, "%02x ", (data = _rbfs_io_read(rbfs->driver, memory_address)));
00101         fflush(fp);
00102         count++;
00103         count2++;
00104     }
00105     fclose(fp);
00106 }
```

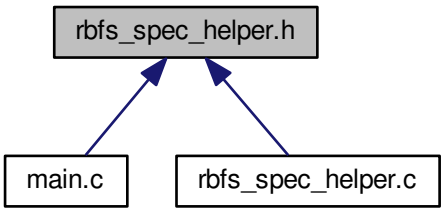
4.19 rbfs_spec_helper.h File Reference

```
#include <rbfs.h>
```

Include dependency graph for rbfs_spec_helper.h:



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



Functions

- void resource_dump (rbfs_resource_t *resource)
- void format_all ()
- char * itob (int i)
- void rbfs_io_memory_dump (rbfs_t *rbfs)

4.19.1 Function Documentation

4.19.1.1 void format_all ( )

Definition at line 20 of file rbfs_spec_helper.c.

4.19.1.2 char* itob (int *i*)

Definition at line 27 of file rbfs_spec_helper.c.

4.19.1.3 void rbfs_io_memory_dump (rbfs_t * *rbfs*)

Definition at line 41 of file rbfs_spec_helper.c.

4.19.1.4 void resource_dump (rbfs_resource_t * *resource*)

Definition at line 7 of file rbfs_spec_helper.c.

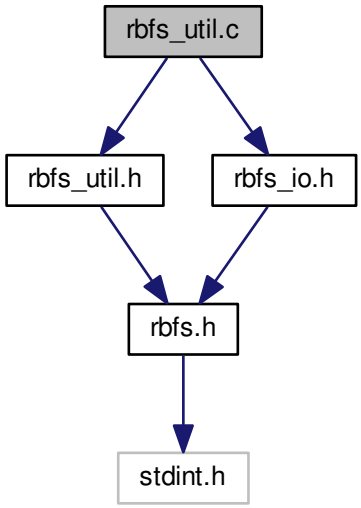
4.20 rbfs_spec_helper.h

```
00001 #ifndef __RBFS_SPEC_HELPER_H__
00002 #define __RBFS_SPEC_HELPER_H__ 1
00003
00004 #include <rbfs.h>
00005
00006 void resource_dump(rbfs_resource_t *resource);
00007 void format_all();
00008 char* itob(int i);
00009 void rbfs_io_memory_dump(rbfs_t *rbfs);
00010
00011 #endif // __RBFS_SPEC_HELPER_H__
```

4.21 rbfs_util.c File Reference

```
#include "rbfs_util.h"
#include "rbfs_io.h"
```

Include dependency graph for rbfs_util.c:



Macros

- #define __RBFS_UTIL_C__ 1

Functions

- void _rbfs_write_rbfs_to_disk (rbfs_driver_t driver, rbfs_t *rbfs)

- void _rbfs_read_rbfs_from_disk (rbfs_driver_t driver, rbfs_t *rbfs)
- rbfs_memory_address_t _rbfs_alloc_cluster (rbfs_t *rbfs)
- uint8_t _rbfs_is_free_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_format_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_free_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_create_cluster_chain (rbfs_t *rbfs, rbfs_cluster_t prev_cluster, rbfs_cluster_t next_cluster)
- void _rbfs_check_for_eor_reached (rbfs_resource_t *resource)
- uint8_t _rbfs_is_eor_reached (rbfs_resource_t *resource)
- uint8_t _rbfs_check_for_availability (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t _rbfs_move_current_position_ahead (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset)
- uint8_t _rbfs_move_current_position_back (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset)
- void _rbfs_format_resorce_descriptor (rbfs_t *rbfs, rbfs_resource_descriptor_t resource_descriptor)
- uint8_t _rbfs_is_driver_monted (rbfs_driver_t driver)
- void _rbfs_set_driver_monted (rbfs_driver_t driver, uint8_t is)
- void _rbfs_free_resource_descriptors (rbfs_t *rbfs)
- void _rbfs_free_resource_descriptor (rbfs_t *rbfs, rbfs_resource_descriptor_t resource_descriptor)
- void _rbfs_format_resource_clusters (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t _rbfs_format_clusterbfs_chain (rbfs_t *rbfs, rbfs_cluster_t cluster)
- uint8_t _rbfs_has_invalid_attributes (rbfs_t *rbfs)

4.21.1 Macro Definition Documentation

4.21.1.1 #define __RBFS_UTIL_C__ 1

rbfs - Simple Resource Based File System

rbfs_util.c

Util lib for rbfs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file rbfs_util.c.

4.21.2 Function Documentation

4.21.2.1 rbfs_memory_address_t _rbfs_alloc_cluster (rbfs_t * *rbfs*)

Allocate a free cluster from disk if any.

Parameters

rbfs	

Returns

Definition at line 37 of file rbfs_util.c.

4.21.2.2 uint8_t _rbfs_check_for_availability (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Check if we are at the end of resource, if yes alloc another cluster and manage the new pointers.

Parameters

rbfs	
resource	

Returns

Definition at line 92 of file rbfs_util.c.

4.21.2.3 void _rbfs_check_for_eor_reached (rbfs_resource_t * *resource*)

Check if the end-of-resource is reached and set or clear the respecitve flag.

Parameters

resource	

Definition at line 80 of file rbfs_util.c.

4.21.2.4 void _rbfs_create_cluster_chain (rbfs_t * *rbfs,* rbfs_cluster_t *prev_cluster,* rbfs_cluster_t *next_cluster*)

Create a chain between two clusters.

Parameters

rbfs	
prev_cluster	
next_cluster	

Definition at line 68 of file rbfs_util.c.

4.21.2.5 void _rbfs_format_cluster (rbfs_t * *rbfs,* rbfs_cluster_t *cluster*)

Format a given cluster.

Parameters

rbfs	
cluster	

Definition at line 56 of file rbfs_util.c.

4.21.2.6 uint8_t _rbfs_format_clusterbfs_chain (rbfs_t * *rbfs,* rbfs_cluster_t *cluster*)

Format a chain of clusters.

Parameters

rbfs	
cluster	

Returns

Definition at line 197 of file rbfs_util.c.

4.21.2.7 void _rbfs_format_resorce_descriptor (rbfs_t * *rbfs,* rbfs_resource_descriptor_t *resource_descriptor*)

Free a resource description.

Parameters

rbfs	
resource_descriptor	

Definition at line 154 of file rbfs_util.c.

4.21.2.8 void _rbfs_format_resource_clusters (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Free resource cluster.

Parameters

rbfs	
resource	

Definition at line 191 of file rbfs_util.c.

4.21.2.9 void _rbfs_free_cluster (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Free a given cluster.

Parameters

rbfs	
cluster	

Definition at line 63 of file rbfs_util.c.

4.21.2.10 void _rbfs_free_resource_descriptor (rbfs_t * *rbfs*, rbfs_resource_descriptor_t *resource_descriptor*)

Close a single resources.

Parameters

rbfs	
resource_descriptor	

Definition at line 182 of file rbfs_util.c.

4.21.2.11 void _rbfs_free_resource_descriptors (rbfs_t * *rbfs*)

Close all resources.

Parameters

rbfs	

Definition at line 175 of file rbfs_util.c.

4.21.2.12 uint8_t _rbfs_has_invalid_attributes (rbfs_t * *rbfs*)

Calculates and evaluate the rbfs attributes.

Parameters

rbfs	

Returns

Definition at line 212 of file rbfs_util.c.

4.21.2.13 uint8_t _rbfs_is_driver_monted (rbfs_driver_t *driver*)

Test if given driver is mouted.

Parameters

driver	

Returns

Definition at line 163 of file rbfs_util.c.

4.21.2.14 uint8_t _rbfs_is_eor_reached (rbfs_resource_t * *resource*)

Test the end-of-resource flag.

Parameters

resource	

Returns

Definition at line 88 of file rbfs_util.c.

4.21.2.15 uint8_t _rbfs_is_free_cluster (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Test if the given cluster is free.

Parameters

rbfs	
cluster	

Returns

Definition at line 51 of file rbfs_util.c.

4.21.2.16 uint8_t _rbfs_move_current_position_ahead (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_int_t *offset*)

Move the current position ahead 'offset' bytes.

Parameters

rbfs	
resource	
offset	

Returns

Definition at line 113 of file rbfs_util.c.

4.21.2.17 uint8_t _rbfs_move_current_position_back (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_int_t *offset*)

Move the current position back 'offset' bytes.

Parameters

rbfs	
resource	
offset	

Returns

Definition at line 132 of file rbfs_util.c.

4.21.2.18 void _rbfs_read_rbfs_from_disk (rbfs_driver_t *driver*, rbfs_t * *rbfs*)

Read a resource system table from disk.

Parameters

driver	
rbfs	

Definition at line 27 of file rbfs_util.c.

4.21.2.19 void _rbfs_set_driver_monted (rbfs_driver_t *driver*, uint8_t *is*)

Set/clear given driver as mouted.

Parameters

driver	
is	

Definition at line 167 of file rbfs_util.c.

4.21.2.20 void _rbfs_write_rbfs_to_disk (rbfs_driver_t *driver*, rbfs_t * *rbfs*)

rbfs - Simple Resource Based File System

rbfs_util.h

Util lib for rbfs

Author

Dalmir da Silva dalmirdasilva@gmail.com Write a resource system table to disk

Parameters

driver	
rbfs	

Definition at line 17 of file rbfs_util.c.

4.22 rbfs_util.c

```
00001
00011 #ifndef __RBFS_UTIL_C__
00012 #define __RBFS_UTIL_C__ 1
00013
00014 #include "rbfs_util.h"
00015 #include "rbfs_io.h"
00016
00017 void _rbfs_write_rbfs_to_disk(rbfs_driver_t driver,
      rbfs_t *rbfs) {
00018     uint8_t i;
00019     uint8_t *p;
00020     rbfs_memory_address_t address =
```

```
      RBFS_FIRST_ADDRESS_OF_MEMORY;
00021     p = (uint8_t *) rbfs;
00022     for (i = 0; i < sizeof (rbfs_t); i++) {
00023         _rbfs_io_write(driver, address++, *(p++));
00024     }
00025 }
00026
00027 void _rbfs_read_rbfs_from_disk(rbfs_driver_t driver,
      rbfs_t *rbfs) {
00028     uint8_t i;
00029     uint8_t *p;
00030     rbfs_memory_address_t address =
      RBFS_FIRST_ADDRESS_OF_MEMORY;
00031     p = (uint8_t *) rbfs;
00032     for (i = 0; i < sizeof (rbfs_t); i++) {
00033         *(p++) = _rbfs_io_read(driver, address++);
00034     }
00035 }
00036
00037 rbfs_memory_address_t _rbfs_alloc_cluster(
      rbfs_t *rbfs) {
00038     rbfs_memory_address_t address;
00039     uint8_t i;
00040     address = rbfs->cluster_table_address;
00041     for (i = 0; i < rbfs->cluster_count; i++) {
00042         if (_rbfs_is_free_cluster(rbfs, (rbfs_cluster_t) i)) {
00043             _rbfs_decrease_free_clusters(rbfs, 1);
00044             return address;
00045         }
00046         address += rbfs->sizeof_cluster;
00047     }
00048     return RBFS_NULL_CLUSTER_ADDRESS;
00049 }
00050
00051 uint8_t _rbfs_is_free_cluster(rbfs_t *rbfs,
      rbfs_cluster_t cluster) {
00052     return (cluster == _rbfs_prev_cluster_by_cluster(rbfs, cluster)) \
00053         && (cluster == _rbfs_next_cluster_by_cluster(rbfs, cluster));
00054 }
00055
00056 void _rbfs_format_cluster(rbfs_t *rbfs, rbfs_cluster_t cluster) {
00057     rbfs_memory_address_t address;
00058     address = _rbfs_cluster_to_address(rbfs, cluster);
00059     _rbfs_io_write(rbfs->driver, CLUSTER_ADDRESS_TO_NEXT(address
      ), cluster);
00060     _rbfs_io_write(rbfs->driver, CLUSTER_ADDRESS_TO_PREV(address
      ), cluster);
00061 }
00062
00063 void _rbfs_free_cluster(rbfs_t *rbfs, rbfs_cluster_t cluster) {
00064     _rbfs_format_cluster(rbfs, cluster);
00065     _rbfs_increase_free_clusters(rbfs, 1);
00066 }
00067
00068 void _rbfs_create_cluster_chain(rbfs_t *rbfs,
      rbfs_cluster_t prev_cluster, rbfs_cluster_t next_cluster) {
00069     rbfs_memory_address_t address;
00070     if (prev_cluster != RBFS_INEXISTENT_CLUSTER) {
00071         address = _rbfs_cluster_to_address(rbfs, prev_cluster);
00072         _rbfs_io_write(rbfs->driver, CLUSTER_ADDRESS_TO_NEXT(
      address), (uint8_t) next_cluster);
00073     }
00074     if (next_cluster != RBFS_INEXISTENT_CLUSTER) {
00075         address = _rbfs_cluster_to_address(rbfs, next_cluster);
00076         _rbfs_io_write(rbfs->driver, CLUSTER_ADDRESS_TO_PREV(
      address), (uint8_t) prev_cluster);
00077     }
00078 }
00079
00080 void _rbfs_check_for_eor_reached(rbfs_resource_t *resource) {
00081     if (resource->current_position >= resource->size) {
00082         resource->flags |= RBFS_RESOURCE_FLAG_BIT_EOR_REACHED;
00083     } else {
00084         resource->flags &= ~RBFS_RESOURCE_FLAG_BIT_EOR_REACHED;
00085     }
00086 }
00087
00088 uint8_t _rbfs_is_eor_reached(rbfs_resource_t *resource) {
00089     return resource->flags & RBFS_RESOURCE_FLAG_BIT_EOR_REACHED;
00090 }
00091
00092 uint8_t _rbfs_check_for_availability(rbfs_t *rbfs,
      rbfs_resource_t *resource) {
00093     rbfs_memory_address_t address;
00094     rbfs_cluster_t cluster;
00095     _rbfs_check_for_eor_reached(resource);
00096     if (resource->cluster_offset >= rbfs->sizeof_cluster) {
```

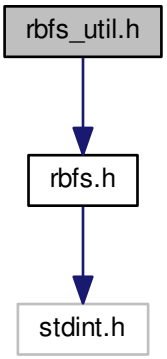
```
00097         if (rbfs_eor(resource)) {
00098             address = _rbfs_alloc_cluster(rbfs);
00099             if (address == RBFS_NULL_CLUSTER_ADDRESS) {
00100                 return 0;
00101             }
00102             cluster = _rbfs_address_to_cluster(rbfs, address);
00103             _rbfs_create_cluster_chain(rbfs, resource->
      current_cluster, cluster);
00104             resource->current_cluster = cluster;
00105         } else {
00106             resource->current_cluster =
      _rbfs_next_cluster_by_cluster(rbfs, resource->
      current_cluster);
00107         }
00108         resource->cluster_offset = rbfs->sizeof_cluster_control;
00109     }
00110     return 1;
00111 }
00112
00113 uint8_t _rbfs_move_current_position_ahead(
      rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset) {
00114     uint8_t until_the_end;
00115     uint8_t how_many_clustes_ahead;
00116     uint8_t i;
00117     resource->current_position += offset;
00118     until_the_end = (rbfs->sizeof_cluster - resource->
      cluster_offset);
00119     if (offset <= until_the_end) {
00120         resource->cluster_offset += offset;
00121         return 1;
00122     }
00123     offset -= until_the_end;
00124     how_many_clustes_ahead = (offset / rbfs->sizeof_cluster_data) + 1;
00125     resource->cluster_offset = (offset % rbfs->sizeof_cluster_data) + rbfs
      ->sizeof_cluster_control;
00126     for (i = 0; i < how_many_clustes_ahead; i++) {
00127         resource->current_cluster = _rbfs_next_cluster_by_cluster
      (rbfs, resource->current_cluster);
00128     }
00129     return 1;
00130 }
00131
00132 uint8_t _rbfs_move_current_position_back(rbfs_t *rbfs,
      rbfs_resource_t *resource, rbfs_seek_int_t offset) {
00133     uint8_t until_the_begin;
00134     uint8_t how_many_clustes_back;
00135     uint8_t i;
00136     resource->current_position -= offset;
00137     until_the_begin = (resource->cluster_offset - rbfs->
      sizeof_cluster_control);
00138     if (offset <= until_the_begin) {
00139         resource->cluster_offset -= offset;
00140         return 1;
00141     }
00142     offset -= until_the_begin;
00143     how_many_clustes_back = (offset / rbfs->sizeof_cluster_data);
00144     if ((offset % rbfs->sizeof_cluster_data) != 0) {
00145         how_many_clustes_back++;
00146     }
00147     resource->cluster_offset = rbfs->sizeof_cluster - (offset % rbfs->
      sizeof_cluster_data);
00148     for (i = 0; i < how_many_clustes_back; i++) {
00149         resource->current_cluster = _rbfs_prev_cluster_by_cluster
      (rbfs, resource->current_cluster);
00150     }
00151     return 1;
00152 }
00153
00154 void _rbfs_format_resorce_descriptor(rbfs_t *rbfs,
      rbfs_resource_descriptor_t resource_descriptor) {
00155     int i;
00156     rbfs_memory_address_t address;
00157     address = _rbfs_resource_descriptor_to_address(rbfs,
      resource_descriptor);
00158     for (i = 0; i < rbfs->sizeof_resource_descriptor; i++) {
00159         _rbfs_io_write(rbfs->driver, address + i, 0x00);
00160     }
00161 }
00162
00163 uint8_t _rbfs_is_driver_monted(rbfs_driver_t driver) {
00164     return rbfs_global_flags.driver_mouted & (1 << driver);
00165 }
00166
00167 void _rbfs_set_driver_monted(rbfs_driver_t driver, uint8_t is) {
00168     if (is) {
00169         rbfs_global_flags.driver_mouted |= (1 << driver);
00170     } else {
```

```
00171         rbfs_global_flags.driver_mouted &= ~(1 << driver);
00172     }
00173 }
00174
00175 void _rbfs_free_resource_descriptors(rbfs_t *rbfs) {
00176     uint8_t i;
00177     for (i = 0; i < rbfs->resource_descriptor_count; i++) {
00178         _rbfs_free_resource_descriptor(rbfs, i);
00179     }
00180 }
00181
00182 void _rbfs_free_resource_descriptor(rbfs_t *rbfs,
      rbfs_resource_descriptor_t resource_descriptor) {
00183     rbfs_memory_address_t address;
00184     uint8_t flags;
00185     address = _rbfs_resource_descriptor_to_address(rbfs,
      resource_descriptor);
00186     flags = _rbfs_io_read(rbfs->driver, RD_ADDRESS_TO_FLAG(address));
00187     flags &= ~(RBFS_RESOURCE_FLAG_BIT_OPENED |
      RBFS_RESOURCE_FLAG_BIT_READ_ONLY);
00188     _rbfs_io_write(rbfs->driver, RD_ADDRESS_TO_FLAG(address), flags);
00189 }
00190
00191 void _rbfs_format_resource_clusters(rbfs_t *rbfs,
      rbfs_resource_t *resource) {
00192     uint8_t freed_clusters;
00193     freed_clusters = _rbfs_format_clusterbfs_chain(rbfs, resource->
      first_cluster);
00194     _rbfs_increase_free_clusters(rbfs, freed_clusters);
00195 }
00196
00197 uint8_t _rbfs_format_clusterbfs_chain(rbfs_t *rbfs,
      rbfs_cluster_t cluster) {
00198     rbfs_cluster_t next_cluster;
00199     uint8_t formated_clusters = 0;
00200     do {
00201         next_cluster = _rbfs_next_cluster_by_cluster(rbfs, cluster);
00202         _rbfs_format_cluster(rbfs, cluster);
00203         formated_clusters++;
00204         if (next_cluster == RBFS_INEXISTENT_CLUSTER || next_cluster == cluster) {
00205             break;
00206         }
00207         cluster = next_cluster;
00208     } while (1);
00209     return formated_clusters;
00210 }
00211
00212 uint8_t _rbfs_has_invalid_attributes(rbfs_t *rbfs) {
00213     if (rbfs->sizeof_resource_descriptor_table != (rbfs->
      sizeof_resource_descriptor * rbfs->
      resource_descriptor_count)) {
00214         // TODO: Use macros or constants here.
00215         return 1;
00216     }
00217     if (rbfs->sizeof_cluster_table != (rbfs->sizeof_cluster * rbfs->
      cluster_count)) {
00218         return 2;
00219     }
00220     if (rbfs->sizeof_cluster != (rbfs->sizeof_cluster_control + rbfs->
      sizeof_cluster_data)) {
00221         return 3;
00222     }
00223     if (rbfs->memory_size != rbfs->sizeof_cluster_table + rbfs->
      cluster_table_address) {
00224         return 4;
00225     }
00226     return 0;
00227 }
00228
00229 #endif // __RBFS_UTIL_C__
```

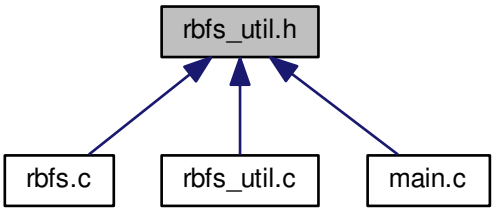
4.23 rbfs_util.h File Reference

```
#include "rbfs.h"
```

Include dependency graph for rbfs_util.h:



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



Macros

- #define _rbfs_resource_code_to_resource_descriptor(resource_code) (rbfs_resource_descriptor_t)(resource_code)
- #define _rbfs_resource_descriptor_to_resource_code(resource_descriptor) (rbfs_resource_code_t)(resource_descriptor)
- #define _rbfs_cluster_to_address(rbfs, cluster) (rbfs_memory_address_t)(rbfs->cluster_table_address + (cluster * rbfs->sizeof_cluster))
- #define _rbfs_address_to_cluster(rbfs, address) (rbfs_cluster_t)((address - rbfs->cluster_table_address) / rbfs->sizeof_cluster)
- #define _rbfs_resource_descriptor_to_address(rbfs, resource_descriptor) (rbfs_memory_address_t)((resource_descriptor * rbfs->sizeof_resource_descriptor) + rbfs->resource_descriptor_table_address)
- #define _rbfs_address_to_resource_descriptor(rbfs, address) (rbfs_resource_descriptor_t)((address - rbfs->resource_descriptor_table_address) / rbfs->sizeof_resource_descriptor)
- #define _rbfs_decrease_free_clusters(rbfs, n)
- #define _rbfs_increase_free_clusters(rbfs, n)
- #define _rbfs_prev_cluster_by_cluster(rbfs, cluster) _rbfs_prev_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))

- #define _rbfs_next_cluster_by_cluster(rbfs, cluster) _rbfs_next_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))
- #define _rbfs_prev_cluster_by_cluster_address(rbfs, address) (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_PREV(address)))
- #define _rbfs_next_cluster_by_cluster_address(rbfs, address) (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_NEXT(address)))

Functions

- void _rbfs_write_rbfs_to_disk (rbfs_driver_t driver, rbfs_t *rbfs)
- void _rbfs_read_rbfs_from_disk (rbfs_driver_t driver, rbfs_t *rbfs)
- rbfs_memory_address_t _rbfs_alloc_cluster (rbfs_t *rbfs)
- uint8_t _rbfs_is_free_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_format_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_free_cluster (rbfs_t *rbfs, rbfs_cluster_t cluster)
- void _rbfs_create_cluster_chain (rbfs_t *rbfs, rbfs_cluster_t prev_cluster, rbfs_cluster_t next_cluster)
- void _rbfs_check_for_eor_reached (rbfs_resource_t *resource)
- uint8_t _rbfs_is_eor_reached (rbfs_resource_t *resource)
- uint8_t _rbfs_check_for_availability (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t _rbfs_move_current_position_ahead (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset)
- uint8_t _rbfs_move_current_position_back (rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset)
- void _rbfs_format_resorce_descriptor (rbfs_t *rbfs, rbfs_resource_descriptor_t resource_descriptor)
- uint8_t _rbfs_is_driver_monted (rbfs_driver_t driver)
- void _rbfs_set_driver_monted (rbfs_driver_t driver, uint8_t is)
- void _rbfs_free_resource_descriptors (rbfs_t *rbfs)
- void _rbfs_free_resource_descriptor (rbfs_t *rbfs, rbfs_resource_descriptor_t resource_descriptor)
- void _rbfs_format_resource_clusters (rbfs_t *rbfs, rbfs_resource_t *resource)
- uint8_t _rbfs_format_clusterbfs_chain (rbfs_t *rbfs, rbfs_cluster_t cluster)
- uint8_t _rbfs_has_invalid_attributes (rbfs_t *rbfs)

4.23.1 Macro Definition Documentation

4.23.1.1 #define _rbfs_address_to_cluster(*rbfs, address*) (rbfs_cluster_t)((address - rbfs->cluster_table_address) / rbfs->sizeof_cluster)

Convert address to cluster.

Parameters

resource	

Definition at line 100 of file rbfs_util.h.

4.23.1.2 #define _rbfs_address_to_resource_descriptor(*rbfs, address*) (rbfs_resource_descriptor_t)((address - rbfs->resource_descriptor_table_address) / rbfs->sizeof_resource_descriptor)

Convert address to rd.

Parameters

resource	

Definition at line 114 of file rbfs_util.h.

4.23.1.3 #define _rbfs_cluster_to_address(*rbfs, cluster*) (rbfs_memory_address_t)(rbfs->cluster_table_address + (cluster * rbfs->sizeof_cluster))

Convert cluster to address.

Parameters

resource	

Definition at line 93 of file rbfs_util.h.

4.23.1.4 #define _rbfs_decrease_free_clusters(*rbfs, n*)

Value:

```
{ \
                                                rbfs->free_clusters -= n; \
                                                _rbfs_write_rbfs_to_disk
      (rbfs->driver, rbfs); \
                                            }
```

Decrease free cluster.

Parameters

rbfs	
resource	

Definition at line 206 of file rbfs_util.h.

4.23.1.5 #define _rbfs_increase_free_clusters(*rbfs, n*)

Value:

```
{ \
                                                rbfs->free_clusters += n; \
                                                _rbfs_write_rbfs_to_disk
      (rbfs->driver, rbfs); \
                                            }
```

Increase free cluster.

Parameters

rbfs	
resource	

Definition at line 217 of file rbfs_util.h.

4.23.1.6 #define _rbfs_next_cluster_by_cluster(*rbfs, cluster*) _rbfs_next_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))

Get the next cluster by a cluster.

Parameters

rbfs	

Returns

Definition at line 253 of file rbfs_util.h.

4.23.1.7 #define _rbfs_next_cluster_by_cluster_address(*rbfs, address*) (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_NEXT(address)))

Get the next cluster by a cluster address.

Parameters

rbfs	

Returns

Definition at line 269 of file rbfs_util.h.

4.23.1.8 #define _rbfs_prev_cluster_by_cluster(*rbfs, cluster*) _rbfs_prev_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))

Get the previous cluster by a cluster.

Parameters

rbfs	

Returns

Definition at line 245 of file rbfs_util.h.

4.23.1.9 #define _rbfs_prev_cluster_by_cluster_address(*rbfs, address*) (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_PREV(address)))

Get the previous cluster by a cluster address.

Parameters

rbfs	

Returns

Definition at line 261 of file rbfs_util.h.

4.23.1.10 #define _rbfs_resource_code_to_resource_descriptor(*resource_code*) (rbfs_resource_descriptor_t)(resource_code)

Convert resource code to rd.

Parameters

resource	

Definition at line 79 of file rbfs_util.h.

4.23.1.11 #define _rbfs_resource_descriptor_to_address(*rbfs, resource_descriptor*) (rbfs_memory_address_t)((resource_descriptor * rbfs->sizeof_resource_descriptor) + rbfs->resource_descriptor_table_address)

Convert rd to address.

Parameters

resource	

Definition at line 107 of file rbfs_util.h.

4.23.1.12 #define _rbfs_resource_descriptor_to_resource_code(*resource_descriptor*) (rbfs_resource_code_↩ t)(resource_descriptor)

Convert rd to resource code.

Parameters

resource	

Definition at line 86 of file rbfs_util.h.

4.23.2 Function Documentation

4.23.2.1 rbfs_memory_address_t _rbfs_alloc_cluster (rbfs_t * *rbfs*)

Allocate a free cluster from disk if any.

Parameters

rbfs	

Returns

Definition at line 37 of file rbfs_util.c.

4.23.2.2 uint8_t _rbfs_check_for_availability (rbfs_t * *rbfs,* rbfs_resource_t * *resource*)

Check if we are at the end of resource, if yes alloc another cluster and manage the new pointers.

Parameters

rbfs	
resource	

Returns

Definition at line 92 of file rbfs_util.c.

4.23.2.3 void _rbfs_check_for_eor_reached (rbfs_resource_t * *resource*)

Check if the end-of-resource is reached and set or clear the respecitve flag.

Parameters

resource	

Definition at line 80 of file rbfs_util.c.

4.23.2.4 void _rbfs_create_cluster_chain (rbfs_t * *rbfs,* rbfs_cluster_t *prev_cluster,* rbfs_cluster_t *next_cluster*)

Create a chain between two clusters.

Parameters

rbfs	
prev_cluster	
next_cluster	

Definition at line 68 of file rbfs_util.c.

4.23.2.5 void _rbfs_format_cluster (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Format a given cluster.

Parameters

rbfs	
cluster	

Definition at line 56 of file rbfs_util.c.

4.23.2.6 uint8_t _rbfs_format_clusterbfs_chain (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Format a chain of clusters.

Parameters

rbfs	
cluster	

Returns

Definition at line 197 of file rbfs_util.c.

4.23.2.7 void _rbfs_format_resorce_descriptor (rbfs_t * *rbfs*, rbfs_resource_descriptor_t *resource_descriptor*)

Free a resource description.

Parameters

rbfs	
resource_↩ descriptor	

Definition at line 154 of file rbfs_util.c.

4.23.2.8 void _rbfs_format_resource_clusters (rbfs_t * *rbfs*, rbfs_resource_t * *resource*)

Free resource cluster.

Parameters

rbfs	
resource	

Definition at line 191 of file rbfs_util.c.

4.23.2.9 void _rbfs_free_cluster (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Free a given cluster.

Parameters

rbfs	
cluster	

Definition at line 63 of file rbfs_util.c.

4.23.2.10 void _rbfs_free_resource_descriptor (rbfs_t * *rbfs*, rbfs_resource_descriptor_t *resource_descriptor*)

Close a single resources.

Parameters

rbfs	
resource_descriptor	

Definition at line 182 of file rbfs_util.c.

4.23.2.11 void _rbfs_free_resource_descriptors (rbfs_t * *rbfs*)

Close all resources.

Parameters

rbfs	

Definition at line 175 of file rbfs_util.c.

4.23.2.12 uint8_t _rbfs_has_invalid_attributes (rbfs_t * *rbfs*)

Calculates and evaluate the rbfs attributes.

Parameters

rbfs	

Returns

Definition at line 212 of file rbfs_util.c.

4.23.2.13 uint8_t _rbfs_is_driver_monted (rbfs_driver_t *driver*)

Test if given driver is mouted.

Parameters

driver	

Returns

Definition at line 163 of file rbfs_util.c.

4.23.2.14 uint8_t _rbfs_is_eor_reached (rbfs_resource_t * *resource*)

Test the end-of-resource flag.

Parameters

resource	

Returns

Definition at line 88 of file rbfs_util.c.

4.23.2.15 uint8_t _rbfs_is_free_cluster (rbfs_t * *rbfs*, rbfs_cluster_t *cluster*)

Test if the given cluster is free.

Parameters

rbfs	
cluster	

Returns

Definition at line 51 of file rbfs_util.c.

4.23.2.16 uint8_t _rbfs_move_current_position_ahead (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_int_t *offset*)

Move the current position ahead 'offset' bytes.

Parameters

rbfs	
resource	
offset	

Returns

Definition at line 113 of file rbfs_util.c.

4.23.2.17 uint8_t _rbfs_move_current_position_back (rbfs_t * *rbfs*, rbfs_resource_t * *resource*, rbfs_seek_int_t *offset*)

Move the current position back 'offset' bytes.

Parameters

rbfs	
resource	
offset	

Returns

Definition at line 132 of file rbfs_util.c.

4.23.2.18 void _rbfs_read_rbfs_from_disk (rbfs_driver_t *driver*, rbfs_t * *rbfs*)

Read a resource system table from disk.

Parameters

driver	
rbfs	

Definition at line 27 of file rbfs_util.c.

4.23.2.19 void _rbfs_set_driver_monted (rbfs_driver_t *driver*, uint8_t *is*)

Set/clear given driver as mouted.

Parameters

driver	
is	

Definition at line 167 of file rbfs_util.c.

4.23.2.20 void _rbfs_write_rbfs_to_disk (rbfs_driver_t *driver*, rbfs_t * *rbfs*)

rbfs - Simple Resource Based File System

rbfs_util.h

Util lib for rbfs

Author

Dalmir da Silva dalmirdasilva@gmail.com Write a resource system table to disk

Parameters

driver	
rbfs	

Definition at line 17 of file rbfs_util.c.

4.24 rbfs_util.h

```
00001
00011 #ifndef __RBFS_UTIL_H__
00012 #define __RBFS_UTIL_H__ 1
00013
00014 #include "rbfs.h"
00015
00022 void _rbfs_write_rbfs_to_disk(rbfs_driver_t driver,
      rbfs_t *rbfs);
00023
00030 void _rbfs_read_rbfs_from_disk(rbfs_driver_t driver,
      rbfs_t *rbfs);
00031
00038 rbfs_memory_address_t _rbfs_alloc_cluster(
      rbfs_t *rbfs);
00039
00047 uint8_t _rbfs_is_free_cluster(rbfs_t *rbfs,
      rbfs_cluster_t cluster);
00048
00055 void _rbfs_format_cluster(rbfs_t *rbfs, rbfs_cluster_t cluster);
00056
00063 void _rbfs_free_cluster(rbfs_t *rbfs, rbfs_cluster_t cluster);
00064
00072 void _rbfs_create_cluster_chain(rbfs_t *rbfs,
      rbfs_cluster_t prev_cluster, rbfs_cluster_t next_cluster);
00073
00079 #define _rbfs_resource_code_to_resource_descriptor(resource_code)
       (rbfs_resource_descriptor_t)(resource_code)
00080
00086 #define _rbfs_resource_descriptor_to_resource_code(resource_descriptor)
       (rbfs_resource_code_t)(resource_descriptor)
00087
00093 #define _rbfs_cluster_to_address(rbfs, cluster)
       (rbfs_memory_address_t)(rbfs->cluster_table_address + (cluster * rbfs->sizeof_cluster))
00094
00100 #define _rbfs_address_to_cluster(rbfs, address)                              (rbfs_cluster_t)((address -
```

```
      rbfs->cluster_table_address) / rbfs->sizeof_cluster)
00101
00107 #define _rbfs_resource_descriptor_to_address(rbfs, resource_descriptor)
       (rbfs_memory_address_t)((resource_descriptor * rbfs->sizeof_resource_descriptor) + rbfs->resource_descriptor_table_addr
00108
00114 #define _rbfs_address_to_resource_descriptor(rbfs, address)
       (rbfs_resource_descriptor_t)((address - rbfs->resource_descriptor_table_address) / rbfs->sizeof_resource_descriptor)
00115
00121 void _rbfs_check_for_eor_reached(rbfs_resource_t *resource);
00122
00129 uint8_t _rbfs_is_eor_reached(rbfs_resource_t *resource);
00130
00139 uint8_t _rbfs_check_for_availability(rbfs_t *rbfs,
      rbfs_resource_t *resource);
00140
00149 uint8_t _rbfs_move_current_position_ahead(
      rbfs_t *rbfs, rbfs_resource_t *resource, rbfs_seek_int_t offset);
00150
00159 uint8_t _rbfs_move_current_position_back(rbfs_t *rbfs,
      rbfs_resource_t *resource, rbfs_seek_int_t offset);
00160
00167 void _rbfs_format_resorce_descriptor(rbfs_t *rbfs,
      rbfs_resource_descriptor_t resource_descriptor);
00168
00175 uint8_t _rbfs_is_driver_monted(rbfs_driver_t driver);
00176
00183 void _rbfs_set_driver_monted(rbfs_driver_t driver, uint8_t is);
00184
00190 void _rbfs_free_resource_descriptors(rbfs_t *rbfs);
00191
00198 void _rbfs_free_resource_descriptor(rbfs_t *rbfs,
      rbfs_resource_descriptor_t resource_descriptor);
00199
00206 #define _rbfs_decrease_free_clusters(rbfs, n)   { \
00207                                                   rbfs->free_clusters -= n; \
00208                                                   _rbfs_write_rbfs_to_disk(rbfs->driver, rbfs); \
00209                                                 }
00210
00217 #define _rbfs_increase_free_clusters(rbfs, n)   { \
00218                                                   rbfs->free_clusters += n; \
00219                                                   _rbfs_write_rbfs_to_disk(rbfs->driver, rbfs); \
00220                                                 }
00221
00228 void _rbfs_format_resource_clusters(rbfs_t *rbfs,
      rbfs_resource_t *resource);
00229
00237 uint8_t _rbfs_format_clusterbfs_chain(rbfs_t *rbfs,
      rbfs_cluster_t cluster);
00238
00245 #define _rbfs_prev_cluster_by_cluster(rbfs, cluster)
       _rbfs_prev_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))
00246
00253 #define _rbfs_next_cluster_by_cluster(rbfs, cluster)
       _rbfs_next_cluster_by_cluster_address(rbfs, _rbfs_cluster_to_address(rbfs, cluster))
00254
00261 #define _rbfs_prev_cluster_by_cluster_address(rbfs, address)
       (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_PREV(address)))
00262
00269 #define _rbfs_next_cluster_by_cluster_address(rbfs, address)
       (rbfs_cluster_t)(_rbfs_io_read(rbfs->driver, CLUSTER_ADDRESS_TO_NEXT(address)))
00270
00277 uint8_t _rbfs_has_invalid_attributes(rbfs_t *rbfs);
00278
00279 #endif // __RBFS_UTIL_H__
```

Index