

Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:52:22

Contents

1 Hierarchical Index	1
1.1 Class Hierarchy	2
2 Class Index	2
2.1 Class List	2
3 File Index	3
3.1 File List	3
4 Class Documentation	4
4.1 Resource Class Reference	4
4.1.1 Detailed Description	5
4.1.2 Member Enumeration Documentation	5
4.1.3 Member Function Documentation	6
4.2 ResourceIO Class Reference	7
4.2.1 Detailed Description	7
4.2.2 Member Function Documentation	7
4.3 ResourceSystem Class Reference	8
4.3.1 Detailed Description	8
4.3.2 Member Enumeration Documentation	9
4.3.3 Member Function Documentation	9
4.4 rs_global_flags_t Struct Reference	9
4.4.1 Detailed Description	9
4.4.2 Member Data Documentation	9
4.5 rs_resource_t Struct Reference	10
4.5.1 Detailed Description	10
4.5.2 Member Data Documentation	10
4.6 rs_stat_t Struct Reference	10
4.6.1 Detailed Description	11
4.6.2 Member Data Documentation	11
4.7 rs_t Struct Reference	11
4.7.1 Detailed Description	11
4.7.2 Member Data Documentation	11
4.8 SimpleArrayResourceIO Class Reference	12
4.8.1 Detailed Description	14
4.8.2 Constructor & Destructor Documentation	14
4.8.3 Member Function Documentation	14
4.8.4 Member Data Documentation	14
4.9 SimpleExternalEepromResourceIO Class Reference	14

4.9.1	Detailed Description	16
4.9.2	Constructor & Destructor Documentation	16
4.9.3	Member Function Documentation	16
4.9.4	Member Data Documentation	16
4.10	SimpleResource Class Reference	16
4.10.1	Detailed Description	18
4.10.2	Constructor & Destructor Documentation	18
4.10.3	Member Function Documentation	18
4.10.4	Member Data Documentation	20
4.11	SimpleResourceIO Class Reference	20
4.11.1	Detailed Description	22
4.11.2	Constructor & Destructor Documentation	22
4.11.3	Member Function Documentation	22
4.11.4	Member Data Documentation	23
4.12	SimpleResourceSystem Class Reference	24
4.12.1	Detailed Description	25
4.12.2	Constructor & Destructor Documentation	25
4.12.3	Member Function Documentation	25
4.12.4	Member Data Documentation	26
4.13	SimpleVirtualResourceIO Class Reference	26
4.13.1	Detailed Description	27
4.13.2	Constructor & Destructor Documentation	28
4.13.3	Member Function Documentation	28
4.13.4	Member Data Documentation	28
5	File Documentation	28
5.1	main.c File Reference	28
5.1.1	Function Documentation	29
5.2	main.c	29
5.3	main.cpp File Reference	31
5.3.1	Macro Definition Documentation	32
5.3.2	Function Documentation	32
5.4	main.cpp	32
5.5	Resource.h File Reference	35
5.6	Resource.h	36
5.7	ResourceIO.h File Reference	37
5.8	ResourceIO.h	37
5.9	ResourceSystem.h File Reference	37
5.10	ResourceSystem.h	38
5.11	rs.c File Reference	38

5.11.1	Macro Definition Documentation	40
5.11.2	Function Documentation	40
5.11.3	Variable Documentation	42
5.12	rs.c	42
5.13	rs.h File Reference	46
5.13.1	Macro Definition Documentation	48
5.13.2	Typedef Documentation	49
5.13.3	Enumeration Type Documentation	49
5.13.4	Function Documentation	51
5.13.5	Variable Documentation	52
5.14	rs.h	52
5.15	rs_init_partition.c File Reference	55
5.15.1	Macro Definition Documentation	56
5.15.2	Function Documentation	56
5.16	rs_init_partition.c	56
5.17	rs_init_partition.h File Reference	57
5.17.1	Enumeration Type Documentation	58
5.17.2	Function Documentation	59
5.18	rs_init_partition.h	59
5.19	rs_io.c File Reference	60
5.19.1	Macro Definition Documentation	61
5.19.2	Function Documentation	61
5.20	rs_io.c	61
5.21	rs_io.h File Reference	61
5.21.1	Function Documentation	62
5.22	rs_io.h	63
5.23	rs_spec.h File Reference	63
5.23.1	Macro Definition Documentation	64
5.23.2	Function Documentation	65
5.24	rs_spec.h	66
5.25	rs_spec_not_virtual.h File Reference	74
5.25.1	Macro Definition Documentation	76
5.25.2	Function Documentation	77
5.26	rs_spec_not_virtual.h	77
5.27	rs_util.c File Reference	78
5.27.1	Macro Definition Documentation	79
5.27.2	Function Documentation	80
5.28	rs_util.c	85
5.29	rs_util.h File Reference	88
5.29.1	Macro Definition Documentation	89

5.29.2 Function Documentation	93
5.30 rs_util.h	98
5.31 rs_util_spec.h File Reference	99
5.32 rs_util_spec.h	99
5.33 SimpleArrayResourceIO.cpp File Reference	99
5.33.1 Macro Definition Documentation	100
5.34 SimpleArrayResourceIO.cpp	101
5.35 SimpleArrayResourceIO.h File Reference	101
5.36 SimpleArrayResourceIO.h	102
5.37 SimpleExternalEepromResourceIO.cpp File Reference	102
5.37.1 Macro Definition Documentation	103
5.38 SimpleExternalEepromResourceIO.cpp	104
5.39 SimpleExternalEepromResourceIO.h File Reference	104
5.40 SimpleExternalEepromResourceIO.h	105
5.41 SimpleResource.cpp File Reference	105
5.41.1 Macro Definition Documentation	106
5.42 SimpleResource.cpp	107
5.43 SimpleResource.h File Reference	108
5.44 SimpleResource.h	109
5.45 SimpleResourceIO.cpp File Reference	110
5.45.1 Macro Definition Documentation	111
5.46 SimpleResourceIO.cpp	111
5.47 SimpleResourceIO.h File Reference	112
5.47.1 Macro Definition Documentation	112
5.48 SimpleResourceIO.h	113
5.49 SimpleResourceSystem.cpp File Reference	114
5.49.1 Macro Definition Documentation	115
5.50 SimpleResourceSystem.cpp	115
5.51 SimpleResourceSystem.h File Reference	115
5.52 SimpleResourceSystem.h	116
5.53 SimpleVirtualResourceIO.cpp File Reference	117
5.53.1 Macro Definition Documentation	118
5.54 SimpleVirtualResourceIO.cpp	119
5.55 SimpleVirtualResourceIO.h File Reference	119
5.56 SimpleVirtualResourceIO.h	120

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Resource	4
SimpleResource	16
ResourceIO	7
SimpleResourceIO	20
SimpleArrayResourceIO	12
SimpleExternalEepromResourceIO	14
SimpleVirtualResourceIO	26
ResourceSystem	8
SimpleResourceSystem	24
rs_global_flags_t	9
rs_resource_t	10
rs_stat_t	10
rs_t	11

2 Class Index

2.1 Class List

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

Resource	
Arduino - Resource interface	4
ResourceIO	
Arduino - Resource interface	7
ResourceSystem	
Arduino - ResourceSystem interface	8
rs_global_flags_t	9
rs_resource_t	10
rs_stat_t	10
rs_t	11
SimpleArrayResourceIO	
Arduino - A simple resource implementation	12
SimpleExternalEepromResourceIO	
Arduino - A simple resource implementation	14

SimpleResource	
Arduino - A simple resource implementation	16
SimpleResourceIO	20
SimpleResourceSystem	
Arduino - A simple resource implementation	24
SimpleVirtualResourceIO	
Arduino - A simple resource implementation	26

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

main.c	28
main.cpp	31
Resource.h	35
ResourceIO.h	37
ResourceSystem.h	37
rs.c	38
rs.h	46
rs_init_partition.c	55
rs_init_partition.h	57
rs_io.c	60
rs_io.h	61
rs_spec.h	63
rs_spec_not_virtual.h	74
rs_util.c	78
rs_util.h	88
rs_util_spec.h	99
SimpleArrayResourceIO.cpp	99
SimpleArrayResourceIO.h	101
SimpleExternalEepromResourceIO.cpp	102
SimpleExternalEepromResourceIO.h	104
SimpleResource.cpp	105
SimpleResource.h	108

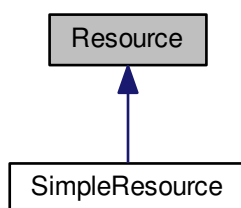
SimpleResourceIO.cpp	110
SimpleResourceIO.h	112
SimpleResourceSystem.cpp	114
SimpleResourceSystem.h	115
SimpleVirtualResourceIO.cpp	117
SimpleVirtualResourceIO.h	119

4 Class Documentation

4.1 Resource Class Reference

```
#include <Resource.h>
```

Inheritance diagram for Resource:



Public Types

- enum [ResourceOperationResult](#) {
[OPERATION_SUCCESS](#) = 0, [OPERATION_ERROR_RESOURCE_OPENED](#) = 1, [OPERATION_ERROR_RESOURCE_CLOSED](#) = 2, [OPERATION_ERROR_RESOURCE_READ_ONLY](#) = 3,
[OPERATION_ERROR_NO_SPACE_AVAILABLE](#) = 4, [OPERATION_ERROR_DRIVER_BUSY](#) = 5, [OPERATION_ERROR_SEEK_OUT_OF_BOUND](#) = 6, [OPERATION_ERROR_RESOURCE_DOES_NOT_ALLOW_READING](#) = 7,
[OPERATION_ERROR_DRIVER_NOT_MOUNTED](#) = 8 }
- enum [OpenOptions](#) { [OPEN_READ_WRITE](#) = 0, [OPEN_READ_ONLY](#) = 1 }
- enum [ResourceSeekOrigin](#) { [SEEK_ORIGIN_BEGIN](#) = 0, [SEEK_ORIGIN_CURRENT](#) = 1 }

Public Member Functions

- virtual bool [open](#) ([OpenOptions](#) options)=0
- virtual bool [close](#) ()=0
- virtual void [write](#) (unsigned char b)=0
- virtual void [writeBytes](#) (unsigned char *buf, int len)=0
- virtual int [read](#) ()=0
- virtual int [readBytes](#) (unsigned char *buf, int len)=0
- virtual bool [seek](#) ([ResourceSeekOrigin](#) origin, unsigned int offset)=0

- virtual bool [truncate](#) ()=0
- virtual void [sync](#) ()=0
- virtual bool [rewind](#) ()=0
- virtual void [release](#) ()=0
- virtual unsigned int [size](#) ()=0
- virtual unsigned int [tell](#) ()=0
- virtual bool [eor](#) ()=0
- virtual bool [error](#) ()=0
- virtual bool [isReadOnly](#) ()=0

4.1.1 Detailed Description

Arduino - [Resource](#) interface.

[Resource.h](#)

This is a resource interface

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file [Resource.h](#).

4.1.2 Member Enumeration Documentation

4.1.2.1 enum Resource::OpenOptions

Enumerator

OPEN_READ_WRITE
OPEN_READ_ONLY

Definition at line 29 of file [Resource.h](#).

4.1.2.2 enum Resource::ResourceOperationResult

Enumerator

OPERATION_SUCCESS
OPERATION_ERROR_RESOURCE_OPENED
OPERATION_ERROR_RESOURCE_CLOSED
OPERATION_ERROR_RESOURCE_READ_ONLY
OPERATION_ERROR_NO_SPACE_AVAILABLE
OPERATION_ERROR_DRIVER_BUSY
OPERATION_ERROR_SEEK_OUT_OF_BOUND
OPERATION_ERROR_RESOURCE_DOES_NOT_ALLOCATED
OPERATION_ERROR_DRIVER_NOT_MOUNTED

Definition at line 17 of file [Resource.h](#).

4.1.2.3 enum Resource::ResourceSeekOrigin

Enumerator

SEEK_ORIGIN_BEGIN
SEEK_ORIGIN_CURRENT

Definition at line 34 of file [Resource.h](#).

4.1.3 Member Function Documentation

4.1.3.1 `virtual bool Resource::close ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.2 `virtual bool Resource::eor ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.3 `virtual bool Resource::error ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.4 `virtual bool Resource::isReadOnly ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.5 `virtual bool Resource::open (OpenOptions options)` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.6 `virtual int Resource::read ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.7 `virtual int Resource::readBytes (unsigned char * buf, int len)` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.8 `virtual void Resource::release ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.9 `virtual bool Resource::rewind ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.10 `virtual bool Resource::seek (ResourceSeekOrigin origin, unsigned int offset)` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.11 `virtual unsigned int Resource::size ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.12 `virtual void Resource::sync ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.13 `virtual unsigned int Resource::tell ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.14 `virtual bool Resource::truncate ()` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.15 `virtual void Resource::write (unsigned char b)` [pure virtual]

Implemented in [SimpleResource](#).

4.1.3.16 `virtual void ResourceIO::writeBytes (unsigned char * buf, int len)` [pure virtual]

Implemented in [SimpleResource](#).

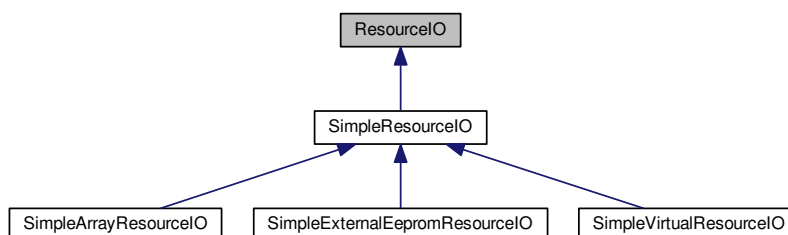
The documentation for this class was generated from the following file:

- [Resource.h](#)

4.2 ResourceIO Class Reference

```
#include <ResourceIO.h>
```

Inheritance diagram for ResourceIO:



Public Member Functions

- virtual bool [open](#) ()=0
- virtual int [read](#) (unsigned int address)=0
- virtual void [write](#) (unsigned int address, unsigned char b)=0
- virtual void [flush](#) ()=0
- virtual void [close](#) ()=0

4.2.1 Detailed Description

Arduino - [Resource](#) interface.

[ResourceIO.h](#)

This is a resource IO interface

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 14 of file [ResourceIO.h](#).

4.2.2 Member Function Documentation

4.2.2.1 `virtual void ResourceIO::close ()` [pure virtual]

Implemented in [SimpleResourceIO](#), and [SimpleVirtualResourceIO](#).

4.2.2.2 `virtual void ResourceIO::flush ()` [pure virtual]

Implemented in [SimpleResourceIO](#), and [SimpleVirtualResourceIO](#).

4.2.2.3 `virtual bool ResourceIO::open () [pure virtual]`

Implemented in [SimpleResourceIO](#), and [SimpleVirtualResourceIO](#).

4.2.2.4 `virtual int ResourceIO::read (unsigned int address) [pure virtual]`

Implemented in [SimpleResourceIO](#).

4.2.2.5 `virtual void ResourceIO::write (unsigned int address, unsigned char b) [pure virtual]`

Implemented in [SimpleResourceIO](#).

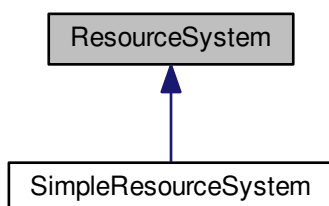
The documentation for this class was generated from the following file:

- [ResourceIO.h](#)

4.3 ResourceSystem Class Reference

```
#include <ResourceSystem.h>
```

Inheritance diagram for ResourceSystem:



Public Types

- enum [MountOptions](#) { [MOUNT_READ_WRITE](#) = 0, [MOUNT_READ_ONLY](#) = 1 }

Public Member Functions

- virtual bool [mount](#) ([MountOptions](#) options)=0
- virtual bool [umount](#) ()=0
- virtual unsigned int [totalSpace](#) ()=0
- virtual unsigned int [availableSpace](#) ()=0

4.3.1 Detailed Description

Arduino - [ResourceSystem](#) interface.

[ResourceSystem.h](#)

This is a resource system interface

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 16 of file [ResourceSystem.h](#).

4.3.2 Member Enumeration Documentation**4.3.2.1 enum ResourceSystem::MountOptions****Enumerator**

MOUNT_READ_WRITE

MOUNT_READ_ONLY

Definition at line 19 of file [ResourceSystem.h](#).

4.3.3 Member Function Documentation

4.3.3.1 virtual unsigned int ResourceSystem::availableSpace () [pure virtual]

Implemented in [SimpleResourceSystem](#).

4.3.3.2 virtual bool ResourceSystem::mount (MountOptions options) [pure virtual]

Implemented in [SimpleResourceSystem](#).

4.3.3.3 virtual unsigned int ResourceSystem::totalSpace () [pure virtual]

Implemented in [SimpleResourceSystem](#).

4.3.3.4 virtual bool ResourceSystem::umount () [pure virtual]

Implemented in [SimpleResourceSystem](#).

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

- [ResourceSystem.h](#)

4.4 rs_global_flags_t Struct Reference

```
#include <rs.h>
```

Public Attributes

- uint8_t [driver_mouted](#)

4.4.1 Detailed Description

Definition at line 142 of file [rs.h](#).

4.4.2 Member Data Documentation

4.4.2.1 uint8_t rs_global_flags_t::driver_mouted

Definition at line 143 of file [rs.h](#).

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

- [rs.h](#)

4.5 `rs_resource_t` Struct Reference

```
#include <rs.h>
```

Public Attributes

- [rs_resource_descriptor_t resource_descriptor](#)
- [rs_cluster_t first_cluster](#)
- [rs_cluster_t current_cluster](#)
- [uint8_t cluster_offset](#)
- [uint16_t size](#)
- [uint16_t current_position](#)
- [uint8_t flags](#)

4.5.1 Detailed Description

Definition at line 132 of file [rs.h](#).

4.5.2 Member Data Documentation

4.5.2.1 `uint8_t rs_resource_t::cluster_offset`

Definition at line 136 of file [rs.h](#).

4.5.2.2 `rs_cluster_t rs_resource_t::current_cluster`

Definition at line 135 of file [rs.h](#).

4.5.2.3 `uint16_t rs_resource_t::current_position`

Definition at line 138 of file [rs.h](#).

4.5.2.4 `rs_cluster_t rs_resource_t::first_cluster`

Definition at line 134 of file [rs.h](#).

4.5.2.5 `uint8_t rs_resource_t::flags`

Definition at line 139 of file [rs.h](#).

4.5.2.6 `rs_resource_descriptor_t rs_resource_t::resource_descriptor`

Definition at line 133 of file [rs.h](#).

4.5.2.7 `uint16_t rs_resource_t::size`

Definition at line 137 of file [rs.h](#).

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

- [rs.h](#)

4.6 `rs_stat_t` Struct Reference

```
#include <rs.h>
```

Public Attributes

- [uint8_t flags](#)

4.6.1 Detailed Description

Definition at line 107 of file [rs.h](#).

4.6.2 Member Data Documentation

4.6.2.1 uint8_t rs_stat_t::flags

Definition at line 108 of file [rs.h](#).

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

- [rs.h](#)

4.7 rs_t Struct Reference

```
#include <rs.h>
```

Public Attributes

- [rs_driver_t driver](#)
- [uint16_t memory_size](#)
- [rs_memory_address_t resource_descriptor_table_address](#)
- [rs_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](#)

4.7.1 Detailed Description

Definition at line 113 of file [rs.h](#).

4.7.2 Member Data Documentation

4.7.2.1 uint8_t rs_t::cluster_count

Definition at line 123 of file [rs.h](#).

4.7.2.2 rs_memory_address_t rs_t::cluster_table_address

Definition at line 117 of file [rs.h](#).

4.7.2.3 `rs_driver_t rs_t::driver`

Definition at line 114 of file [rs.h](#).

4.7.2.4 `uint8_t rs_t::flags`

Definition at line 127 of file [rs.h](#).

4.7.2.5 `uint8_t rs_t::free_clusters`

Definition at line 126 of file [rs.h](#).

4.7.2.6 `uint16_t rs_t::memory_size`

Definition at line 115 of file [rs.h](#).

4.7.2.7 `uint8_t rs_t::resource_descriptor_count`

Definition at line 122 of file [rs.h](#).

4.7.2.8 `rs_memory_address_t rs_t::resource_descriptor_table_address`

Definition at line 116 of file [rs.h](#).

4.7.2.9 `uint8_t rs_t::sizeof_cluster`

Definition at line 121 of file [rs.h](#).

4.7.2.10 `uint8_t rs_t::sizeof_cluster_control`

Definition at line 125 of file [rs.h](#).

4.7.2.11 `uint8_t rs_t::sizeof_cluster_data`

Definition at line 124 of file [rs.h](#).

4.7.2.12 `uint16_t rs_t::sizeof_cluster_table`

Definition at line 119 of file [rs.h](#).

4.7.2.13 `uint8_t rs_t::sizeof_resource_descriptor`

Definition at line 120 of file [rs.h](#).

4.7.2.14 `uint16_t rs_t::sizeof_resource_descriptor_table`

Definition at line 118 of file [rs.h](#).

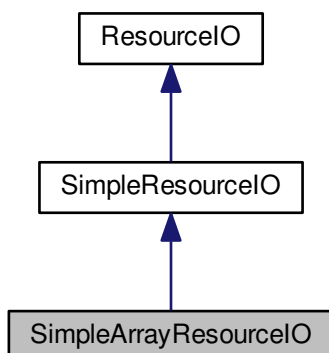
The documentation for this struct was generated from the following file:

- [rs.h](#)

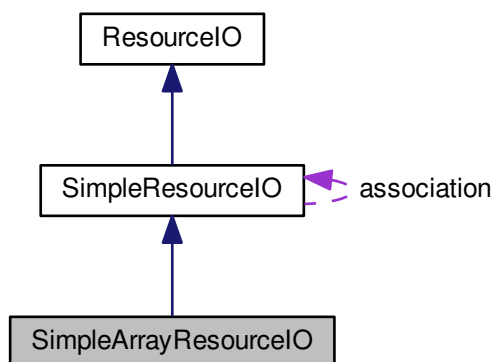
4.8 SimpleArrayResourceIO Class Reference

```
#include <SimpleArrayResourceIO.h>
```


Inheritance diagram for SimpleArrayResourceIO:



Collaboration diagram for SimpleArrayResourceIO:



Public Member Functions

- [SimpleArrayResourceIO](#) (unsigned char *[array](#), unsigned int [size](#))

Protected Member Functions

- virtual int [readBytes](#) (unsigned int address, unsigned char *buf, int len)
- virtual void [writeBytes](#) (unsigned int address, unsigned char *buf, int len)

Private Attributes

- unsigned char * [array](#)
- unsigned int [size](#)

Additional Inherited Members

4.8.1 Detailed Description

Arduino - A simple resource implementation.

[SimpleArrayResourceIO.h](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 16 of file [SimpleArrayResourceIO.h](#).

4.8.2 Constructor & Destructor Documentation

4.8.2.1 SimpleArrayResourceIO::SimpleArrayResourceIO (unsigned char * array, unsigned int size)

Definition at line 16 of file [SimpleArrayResourceIO.cpp](#).

4.8.3 Member Function Documentation

4.8.3.1 int SimpleArrayResourceIO::readBytes (unsigned int address, unsigned char * buf, int len) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 19 of file [SimpleArrayResourceIO.cpp](#).

4.8.3.2 void SimpleArrayResourceIO::writeBytes (unsigned int address, unsigned char * buf, int len) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 31 of file [SimpleArrayResourceIO.cpp](#).

4.8.4 Member Data Documentation

4.8.4.1 unsigned char* SimpleArrayResourceIO::array [private]

Definition at line 18 of file [SimpleArrayResourceIO.h](#).

4.8.4.2 unsigned int SimpleArrayResourceIO::size [private]

Definition at line 19 of file [SimpleArrayResourceIO.h](#).

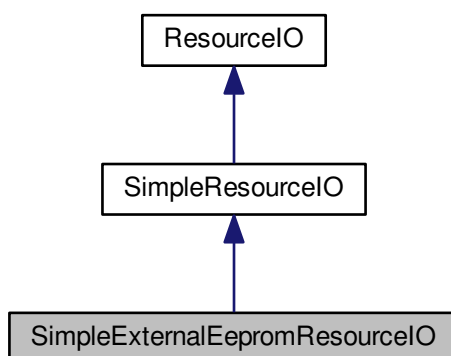
The documentation for this class was generated from the following files:

- [SimpleArrayResourceIO.h](#)
- [SimpleArrayResourceIO.cpp](#)

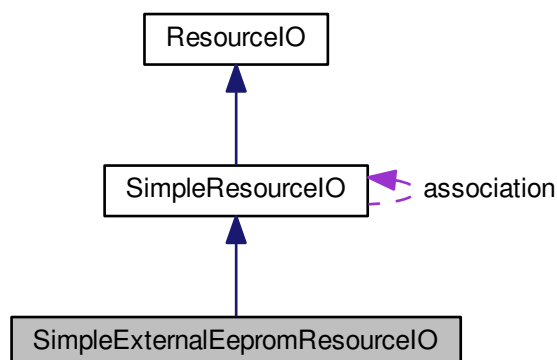
4.9 SimpleExternalEepromResourceIO Class Reference

```
#include <SimpleExternalEepromResourceIO.h>
```

Inheritance diagram for SimpleExternalEepromResourceIO:



Collaboration diagram for SimpleExternalEepromResourceIO:



Public Member Functions

- [SimpleExternalEepromResourceIO](#) (ExternalEeprom *[externalEeprom](#))

Protected Member Functions

- virtual int [readBytes](#) (unsigned int address, unsigned char *buf, int len)
- virtual void [writeBytes](#) (unsigned int address, unsigned char *buf, int len)

Private Attributes

- ExternalEeprom * [externalEeprom](#)

Additional Inherited Members

4.9.1 Detailed Description

Arduino - A simple resource implementation.

SimpleExternalEepromResourceIO.h

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 17 of file [SimpleExternalEepromResourceIO.h](#).

4.9.2 Constructor & Destructor Documentation

4.9.2.1 SimpleExternalEepromResourceIO::SimpleExternalEepromResourceIO (ExternalEeprom * externalEeprom)

Definition at line 16 of file [SimpleExternalEepromResourceIO.cpp](#).

4.9.3 Member Function Documentation

4.9.3.1 int SimpleExternalEepromResourceIO::readBytes (unsigned int address, unsigned char * buf, int len) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 19 of file [SimpleExternalEepromResourceIO.cpp](#).

4.9.3.2 void SimpleExternalEepromResourceIO::writeBytes (unsigned int address, unsigned char * buf, int len) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 23 of file [SimpleExternalEepromResourceIO.cpp](#).

4.9.4 Member Data Documentation

4.9.4.1 ExternalEeprom* SimpleExternalEepromResourceIO::externalEeprom [private]

Definition at line 19 of file [SimpleExternalEepromResourceIO.h](#).

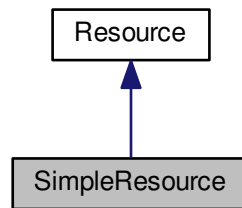
The documentation for this class was generated from the following files:

- [SimpleExternalEepromResourceIO.h](#)
- [SimpleExternalEepromResourceIO.cpp](#)

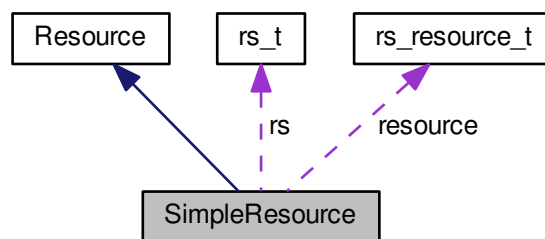
4.10 SimpleResource Class Reference

```
#include <SimpleResource.h>
```

Inheritance diagram for SimpleResource:



Collaboration diagram for SimpleResource:



Public Member Functions

- [SimpleResource](#) ([rs_resource_code_t](#) code, [rs_t](#) *rs)
- [ResourceOperationResult](#) [getLastOperationResult](#) ()
- virtual void [setCode](#) (int code)
- virtual int [getCode](#) ()
- virtual bool [open](#) ([OpenOptions](#) options)
- virtual bool [close](#) ()
- virtual void [write](#) (unsigned char b)
- virtual void [writeBytes](#) (unsigned char *buf, int len)
- virtual int [read](#) ()
- virtual int [readBytes](#) (unsigned char *buf, int len)
- virtual bool [seek](#) ([ResourceSeekOrigin](#) origin, unsigned int offset)
- virtual bool [truncate](#) ()
- virtual void [sync](#) ()
- virtual bool [rewind](#) ()
- virtual void [release](#) ()
- virtual unsigned int [size](#) ()
- virtual unsigned int [tell](#) ()
- virtual bool [eor](#) ()
- virtual bool [error](#) ()
- virtual bool [isReadOnly](#) ()

Private Attributes

- [rs_resource_code_t code](#)
- [rs_resource_t resource](#)
- [rs_t * rs](#)
- [ResourceOperationResult lastOperationResult](#)

Additional Inherited Members

4.10.1 Detailed Description

Arduino - A simple resource implementation.

[SimpleResource.h](#)

This is the [Resource](#) representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 19 of file [SimpleResource.h](#).

4.10.2 Constructor & Destructor Documentation

4.10.2.1 SimpleResource::SimpleResource (rs_resource_code_t code, rs_t * rs)

Definition at line 16 of file [SimpleResource.cpp](#).

4.10.3 Member Function Documentation

4.10.3.1 bool SimpleResource::close () [virtual]

Implements [Resource](#).

Definition at line 25 of file [SimpleResource.cpp](#).

4.10.3.2 bool SimpleResource::eor () [virtual]

Implements [Resource](#).

Definition at line 102 of file [SimpleResource.cpp](#).

4.10.3.3 bool SimpleResource::error () [virtual]

Implements [Resource](#).

Definition at line 106 of file [SimpleResource.cpp](#).

4.10.3.4 virtual int SimpleResource::getCode () [inline],[virtual]

Definition at line 36 of file [SimpleResource.h](#).

4.10.3.5 ResourceOperationResult SimpleResource::getLastOperationResult () [inline]

Definition at line 28 of file [SimpleResource.h](#).

4.10.3.6 bool SimpleResource::isReadOnly () [virtual]

Implements [Resource](#).

Definition at line 110 of file [SimpleResource.cpp](#).

4.10.3.7 `bool SimpleResource::open (OpenOptions options)` [virtual]

Implements [Resource](#).

Definition at line 20 of file [SimpleResource.cpp](#).

4.10.3.8 `int SimpleResource::read ()` [virtual]

Implements [Resource](#).

Definition at line 42 of file [SimpleResource.cpp](#).

4.10.3.9 `int SimpleResource::readBytes (unsigned char * buf, int len)` [virtual]

Implements [Resource](#).

Definition at line 49 of file [SimpleResource.cpp](#).

4.10.3.10 `void SimpleResource::release ()` [virtual]

Implements [Resource](#).

Definition at line 89 of file [SimpleResource.cpp](#).

4.10.3.11 `bool SimpleResource::rewind ()` [virtual]

Implements [Resource](#).

Definition at line 84 of file [SimpleResource.cpp](#).

4.10.3.12 `bool SimpleResource::seek (ResourceSeekOrigin origin, unsigned int offset)` [virtual]

Implements [Resource](#).

Definition at line 69 of file [SimpleResource.cpp](#).

4.10.3.13 `virtual void SimpleResource::setCode (int code)` [inline],[virtual]

Definition at line 32 of file [SimpleResource.h](#).

4.10.3.14 `unsigned int SimpleResource::size ()` [virtual]

Implements [Resource](#).

Definition at line 94 of file [SimpleResource.cpp](#).

4.10.3.15 `void SimpleResource::sync ()` [virtual]

Implements [Resource](#).

Definition at line 79 of file [SimpleResource.cpp](#).

4.10.3.16 `unsigned int SimpleResource::tell ()` [virtual]

Implements [Resource](#).

Definition at line 98 of file [SimpleResource.cpp](#).

4.10.3.17 `bool SimpleResource::truncate ()` [virtual]

Implements [Resource](#).

Definition at line 74 of file [SimpleResource.cpp](#).

4.10.3.18 `void SimpleResource::write (unsigned char b)` [virtual]

Implements [Resource](#).

Definition at line 31 of file [SimpleResource.cpp](#).

4.10.3.19 `void SimpleResource::writeBytes (unsigned char * buf, int len)` [virtual]

Implements [Resource](#).

Definition at line 35 of file [SimpleResource.cpp](#).

4.10.4 Member Data Documentation

4.10.4.1 `rs_resource_code_t SimpleResource::code` [private]

Definition at line 20 of file [SimpleResource.h](#).

4.10.4.2 `ResourceOperationResult SimpleResource::lastOperationResult` [private]

Definition at line 23 of file [SimpleResource.h](#).

4.10.4.3 `rs_resource_t SimpleResource::resource` [private]

Definition at line 21 of file [SimpleResource.h](#).

4.10.4.4 `rs_t* SimpleResource::rs` [private]

Definition at line 22 of file [SimpleResource.h](#).

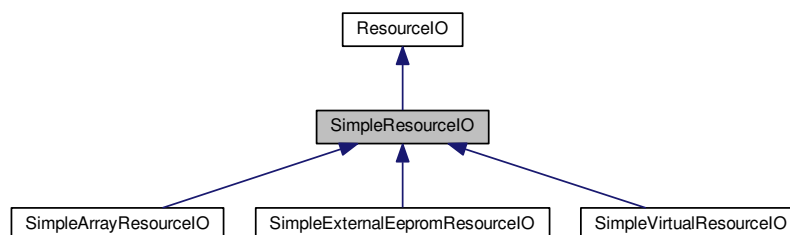
The documentation for this class was generated from the following files:

- [SimpleResource.h](#)
- [SimpleResource.cpp](#)

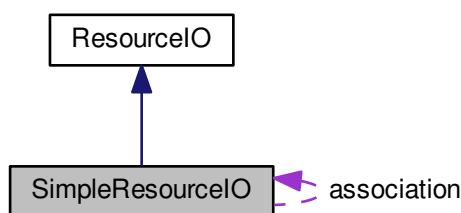
4.11 SimpleResourceIO Class Reference

```
#include <SimpleResourceIO.h>
```

Inheritance diagram for SimpleResourceIO:



Collaboration diagram for SimpleResourceIO:



Public Member Functions

- virtual bool [open](#) ()
- virtual int [read](#) (unsigned int address)
- virtual void [write](#) (unsigned int address, unsigned char b)
- virtual void [flush](#) ()
- virtual void [close](#) ()
- unsigned int [getCacheHit](#) ()
- unsigned int [getCacheMiss](#) ()

Static Public Member Functions

- static void [associateIO](#) ([SimpleResourceIO](#) *io, int driver)
- static [SimpleResourceIO](#) * [getAssociatedIO](#) (int driver)

Protected Member Functions

- [SimpleResourceIO](#) ()
- virtual int [readBytes](#) (unsigned int address, unsigned char *buf, int len)
- virtual void [writeBytes](#) (unsigned int address, unsigned char *buf, int len)

Private Member Functions

- void [checkCache](#) (unsigned int address)

Private Attributes

- bool [wasCacheChanged](#)
- bool [wasCacheInitialized](#)
- unsigned int [cacheMemoryAddress](#)
- unsigned char [cache](#) [[RESOURCE_IO_CACHE_SIZE](#)]
- unsigned int [cacheMiss](#)
- unsigned int [cacheHit](#)
- unsigned int [validCacheSize](#)

Static Private Attributes

- static [SimpleResourceIO](#) * [association](#) [[RESOURCE_IO_DRIVERS_NUM](#)]

4.11.1 Detailed Description

Definition at line 19 of file [SimpleResourceIO.h](#).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 [SimpleResourceIO::SimpleResourceIO](#) () [inline], [protected]

Definition at line 44 of file [SimpleResourceIO.h](#).

4.11.3 Member Function Documentation

4.11.3.1 void [SimpleResourceIO::associateIO](#) ([SimpleResourceIO](#) * *io*, int *driver*) [static]

Definition at line 22 of file [SimpleResourceIO.cpp](#).

4.11.3.2 void [SimpleResourceIO::checkCache](#) (unsigned int *address*) [inline], [private]

Definition at line 29 of file [SimpleResourceIO.h](#).

4.11.3.3 void [SimpleResourceIO::close](#) () [virtual]

Implements [ResourceIO](#).

Reimplemented in [SimpleVirtualResourceIO](#).

Definition at line 50 of file [SimpleResourceIO.cpp](#).

4.11.3.4 void [SimpleResourceIO::flush](#) () [virtual]

Implements [ResourceIO](#).

Reimplemented in [SimpleVirtualResourceIO](#).

Definition at line 44 of file [SimpleResourceIO.cpp](#).

4.11.3.5 [SimpleResourceIO](#) * [SimpleResourceIO::getAssociatedIO](#) (int *driver*) [static]

Definition at line 18 of file [SimpleResourceIO.cpp](#).

4.11.3.6 unsigned int [SimpleResourceIO::getCacheHit](#) () [inline]

Definition at line 75 of file [SimpleResourceIO.h](#).

4.11.3.7 unsigned int [SimpleResourceIO::getCacheMiss](#) () [inline]

Definition at line 79 of file [SimpleResourceIO.h](#).

4.11.3.8 bool [SimpleResourceIO::open](#) () [virtual]

Implements [ResourceIO](#).

Reimplemented in [SimpleVirtualResourceIO](#).

Definition at line 26 of file [SimpleResourceIO.cpp](#).

4.11.3.9 `int SimpleResourceIO::read (unsigned int address) [virtual]`

Implements [ResourceIO](#).

Definition at line 30 of file [SimpleResourceIO.cpp](#).

4.11.3.10 `virtual int SimpleResourceIO::readBytes (unsigned int address, unsigned char * buf, int len) [inline],
[protected], [virtual]`

Reimplemented in [SimpleVirtualResourceIO](#), [SimpleArrayResourceIO](#), and [SimpleExternalEepromResourceIO](#).

Definition at line 53 of file [SimpleResourceIO.h](#).

4.11.3.11 `void SimpleResourceIO::write (unsigned int address, unsigned char b) [virtual]`

Implements [ResourceIO](#).

Definition at line 38 of file [SimpleResourceIO.cpp](#).

4.11.3.12 `virtual void SimpleResourceIO::writeBytes (unsigned int address, unsigned char * buf, int len) [inline],
[protected], [virtual]`

Reimplemented in [SimpleVirtualResourceIO](#), [SimpleArrayResourceIO](#), and [SimpleExternalEepromResourceIO](#).

Definition at line 56 of file [SimpleResourceIO.h](#).

4.11.4 Member Data Documentation

4.11.4.1 `SimpleResourceIO * SimpleResourceIO::association [static], [private]`

Definition at line 22 of file [SimpleResourceIO.h](#).

4.11.4.2 `unsigned char SimpleResourceIO::cache[RESOURCE_IO_CACHE_SIZE] [private]`

Definition at line 25 of file [SimpleResourceIO.h](#).

4.11.4.3 `unsigned int SimpleResourceIO::cacheHit [private]`

Definition at line 26 of file [SimpleResourceIO.h](#).

4.11.4.4 `unsigned int SimpleResourceIO::cacheMemoryAddress [private]`

Definition at line 24 of file [SimpleResourceIO.h](#).

4.11.4.5 `unsigned int SimpleResourceIO::cacheMiss [private]`

Definition at line 26 of file [SimpleResourceIO.h](#).

4.11.4.6 `unsigned int SimpleResourceIO::validCacheSize [private]`

Definition at line 27 of file [SimpleResourceIO.h](#).

4.11.4.7 `bool SimpleResourceIO::wasCacheChanged [private]`

Definition at line 23 of file [SimpleResourceIO.h](#).

4.11.4.8 `bool SimpleResourceIO::wasCacheInitialized [private]`

Definition at line 23 of file [SimpleResourceIO.h](#).

The documentation for this class was generated from the following files:

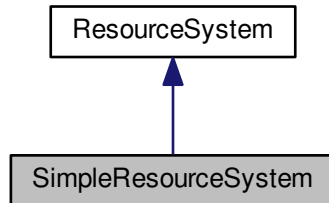
- [SimpleResourceIO.h](#)

- [SimpleResourceIO.cpp](#)

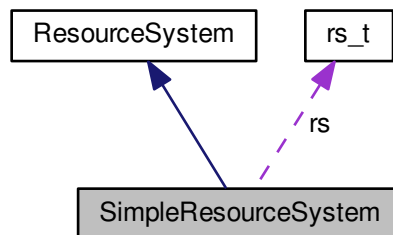
4.12 SimpleResourceSystem Class Reference

```
#include <SimpleResourceSystem.h>
```

Inheritance diagram for SimpleResourceSystem:



Collaboration diagram for SimpleResourceSystem:



Public Member Functions

- [SimpleResourceSystem](#) (int driver)
- [rs_t * getRs \(\)](#)
- [Resource::ResourceOperationResult getLastOperationResult \(\)](#)
- virtual bool [mount](#) ([MountOptions](#) options)
- virtual bool [umount](#) ()
- [SimpleResource alloc](#) ()
- [SimpleResource getResourceByCode](#) (int code)
- virtual unsigned int [totalSpace](#) ()
- virtual unsigned int [availableSpace](#) ()

Static Public Member Functions

- static bool [format](#) ([rs_t *rs](#))

Private Attributes

- [rs_t rs](#)
- [Resource::ResourceOperationResult lastOperationResult](#)

Additional Inherited Members

4.12.1 Detailed Description

Arduino - A simple resource implementation.

[SimpleResourceSystem.h](#)

This is the [Resource](#) system itself.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 19 of file [SimpleResourceSystem.h](#).

4.12.2 Constructor & Destructor Documentation

4.12.2.1 SimpleResourceSystem::SimpleResourceSystem (int *driver*)

Definition at line 16 of file [SimpleResourceSystem.cpp](#).

4.12.3 Member Function Documentation

4.12.3.1 SimpleResource SimpleResourceSystem::alloc ()

Definition at line 32 of file [SimpleResourceSystem.cpp](#).

4.12.3.2 unsigned int SimpleResourceSystem::availableSpace () [virtual]

Implements [ResourceSystem](#).

Definition at line 51 of file [SimpleResourceSystem.cpp](#).

4.12.3.3 static bool SimpleResourceSystem::format (rs_t* *rs*) [inline],[static]

Definition at line 26 of file [SimpleResourceSystem.h](#).

4.12.3.4 Resource::ResourceOperationResult SimpleResourceSystem::getLastOperationResult () [inline]

Definition at line 35 of file [SimpleResourceSystem.h](#).

4.12.3.5 SimpleResource SimpleResourceSystem::getResourceByCode (int *code*)

Definition at line 42 of file [SimpleResourceSystem.cpp](#).

4.12.3.6 rs_t* SimpleResourceSystem::getRs () [inline]

Definition at line 31 of file [SimpleResourceSystem.h](#).

4.12.3.7 bool SimpleResourceSystem::mount (MountOptions *options*) [virtual]

Implements [ResourceSystem](#).

Definition at line 21 of file [SimpleResourceSystem.cpp](#).

4.12.3.8 `unsigned int SimpleResourceSystem::totalSpace () [virtual]`

Implements [ResourceSystem](#).

Definition at line 47 of file [SimpleResourceSystem.cpp](#).

4.12.3.9 `bool SimpleResourceSystem::umount () [virtual]`

Implements [ResourceSystem](#).

Definition at line 26 of file [SimpleResourceSystem.cpp](#).

4.12.4 Member Data Documentation

4.12.4.1 `Resource::ResourceOperationResult SimpleResourceSystem::lastOperationResult [private]`

Definition at line 21 of file [SimpleResourceSystem.h](#).

4.12.4.2 `rs_t SimpleResourceSystem::rs [private]`

Definition at line 20 of file [SimpleResourceSystem.h](#).

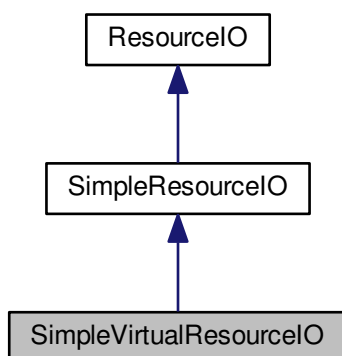
The documentation for this class was generated from the following files:

- [SimpleResourceSystem.h](#)
- [SimpleResourceSystem.cpp](#)

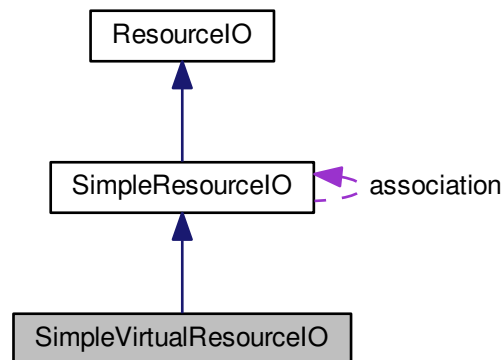
4.13 SimpleVirtualResourceIO Class Reference

```
#include <SimpleVirtualResourceIO.h>
```

Inheritance diagram for SimpleVirtualResourceIO:



Collaboration diagram for SimpleVirtualResourceIO:



Public Member Functions

- [SimpleVirtualResourceIO](#) (char *[fileName](#))
- virtual bool [open](#) ()
- virtual void [flush](#) ()
- virtual void [close](#) ()

Protected Member Functions

- virtual int [readBytes](#) (unsigned int address, unsigned char *buf, int len)
- virtual void [writeBytes](#) (unsigned int address, unsigned char *buf, int len)

Private Attributes

- char * [fileName](#)
- FILE * [fp](#)

Additional Inherited Members

4.13.1 Detailed Description

Arduino - A simple resource implementation.

[SimpleVirtualResourceIO.h](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 17 of file [SimpleVirtualResourceIO.h](#).

4.13.2 Constructor & Destructor Documentation

4.13.2.1 SimpleVirtualResourceIO::SimpleVirtualResourceIO (char * *fileName*)

Definition at line 18 of file [SimpleVirtualResourceIO.cpp](#).

4.13.3 Member Function Documentation

4.13.3.1 void SimpleVirtualResourceIO::close () [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 36 of file [SimpleVirtualResourceIO.cpp](#).

4.13.3.2 void SimpleVirtualResourceIO::flush () [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 31 of file [SimpleVirtualResourceIO.cpp](#).

4.13.3.3 bool SimpleVirtualResourceIO::open () [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 22 of file [SimpleVirtualResourceIO.cpp](#).

4.13.3.4 int SimpleVirtualResourceIO::readBytes (unsigned int *address*, unsigned char * *buf*, int *len*) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 41 of file [SimpleVirtualResourceIO.cpp](#).

4.13.3.5 void SimpleVirtualResourceIO::writeBytes (unsigned int *address*, unsigned char * *buf*, int *len*) [protected], [virtual]

Reimplemented from [SimpleResourceIO](#).

Definition at line 46 of file [SimpleVirtualResourceIO.cpp](#).

4.13.4 Member Data Documentation

4.13.4.1 char* SimpleVirtualResourceIO::fileName [private]

Definition at line 19 of file [SimpleVirtualResourceIO.h](#).

4.13.4.2 FILE* SimpleVirtualResourceIO::fp [private]

Definition at line 20 of file [SimpleVirtualResourceIO.h](#).

The documentation for this class was generated from the following files:

- [SimpleVirtualResourceIO.h](#)
- [SimpleVirtualResourceIO.cpp](#)

5 File Documentation

5.1 main.c File Reference

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

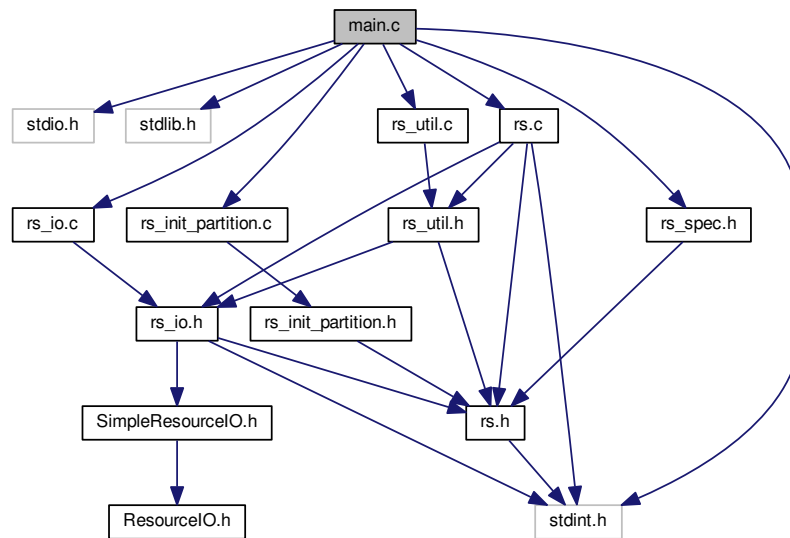


```

#include <stdlib.h>
#include <stdint.h>
#include "rs_io.c"
#include "rs_util.c"
#include "rs.c"
#include "rs_init_partition.c"
#include "rs_spec.h"

```

Include dependency graph for main.c:



Functions

- void [resource_dump](#) ([rs_resource_t](#) *resource)
- void [format_all](#) ()
- int [main](#) ()

5.1.1 Function Documentation

5.1.1.1 void [format_all](#) ()

Definition at line 14 of file [main.c](#).

5.1.1.2 int [main](#) ()

Definition at line 21 of file [main.c](#).

5.1.1.3 void [resource_dump](#) ([rs_resource_t](#) * *resource*)

Definition at line 56 of file [main.c](#).

5.2 main.c

```

00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <stdint.h>
00004

```

```

00005 #include "rs_io.c"
00006 #include "rs_util.c"
00007 #include "rs.c"
00008 #include "rs_init_partition.c"
00009
00010 void resource_dump(rs_resource_t *resource);
00011
00012 #include "rs_spec.h"
00013
00014 void format_all() {
00015     uint16_t i;
00016     for (i = 0; i < 0x7fff; i++) {
00017         _rs_io_write(RS_DRIVER_VIRTUAL, i, 0x00);
00018     }
00019 }
00020
00021 int main() {
00022     rs_t rs;
00023
00024     format_all();
00025     rs_init_partition(&rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00026     rs_format(&rs);
00027     format_spec(&rs);
00028     mount_spec(&rs);
00029     umount_spec(&rs);
00030     alloc_resource_spec(&rs);
00031     try_to_alloc_resources_that_is_possible_spec(&rs);
00032     open_resource_spec(&rs);
00033     write_resource_spec(&rs);
00034     rewind_resource_spec(&rs);
00035     read_resource_spec(&rs);
00036     close_resource_spec(&rs);
00037     try_read_when_end_of_resource_is_reached_spec(&rs);
00038     try_read_when_resource_is_closed_spec(&rs);
00039     seek_resource_spec(&rs);
00040     random_read_resource_spec(&rs);
00041     random_read_with_seek_resource_spec(&rs);
00042     random_read_with_seek_opening_resource_spec(&rs);
00043     size_resource_spec(&rs);
00044     tell_resource_spec(&rs);
00045     tell_with_seek_resource_spec(&rs);
00046     total_space_resource_spec(&rs);
00047     allocating_multi_format_spec(&rs);
00048     read_only_mounting_spec(&rs);
00049     read_only_opening_spec(&rs);
00050
00051     rs_mount(RS_SPEC_DRIVER, &rs, RS_MOUNT_OPTION_NORMAL);
00052     _rs_io_memory_dump(&rs);
00053     return 0;
00054 }
00055
00056 void resource_dump(rs_resource_t *resource) {
00057     printf("==== resource dump begin =====\n");
00058     printf(" resource descriptor: %6d %s\n", resource->resource_descriptor,
itob(resource->resource_descriptor));
00059     printf(" first cluster:_____ %6d %s\n", resource->first_cluster,
itob(resource->first_cluster));
00060     printf(" current cluster:_____ %6d %s\n", resource->current_cluster,
itob(resource->current_cluster));
00061     printf(" cluster offset:_____ %6d %s\n", resource->cluster_offset,
itob(resource->cluster_offset));
00062     printf(" size:_____ %6d %s\n", resource->size, itob(resource->
size));
00063     printf(" current position:_____ %6d %s\n", resource->current_position,
itob(resource->current_position));
00064     printf(" flags:_____ %6d %s\n", resource->flags, itob(resource->
flags));
00065     printf(" errors:_____ %6d %s\n", rs_error(resource), itob(
rs_error(resource)));
00066     printf("==== resource dump end =====\n");
00067 }

```

5.3 main.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <rs.h>
#include <rs_util.h>
#include <rs_io.c>
#include <rs_util.c>
#include <rs.c>
#include <rs_init_partition.c>
#include <Seekable.h>
#include <Seekable.cpp>
#include <Closeable.h>
#include <Closeable.cpp>
#include <InputStream.h>
#include <InputStream.cpp>
#include <ResourceInputStream.h>
#include <ResourceInputStream.cpp>
#include <OutputStream.h>
#include <OutputStream.cpp>
#include <ResourceOutputStream.h>
#include <ResourceOutputStream.cpp>
#include <SimpleResource.h>
#include <SimpleResourceSystem.h>
#include <SimpleResource.cpp>
#include <SimpleResourceSystem.cpp>
#include <SimpleResourceIO.h>
#include <SimpleResourceIO.cpp>
#include <SimpleVirtualResourceIO.h>
#include <SimpleVirtualResourceIO.cpp>
#include <SimpleExternalEepromResourceIO.h>
#include <SimpleExternalEepromResourceIO.cpp>
#include <SimpleArrayResourceIO.h>
#include <SimpleArrayResourceIO.cpp>
#include <ExternalEeprom.h>
#include <ExternalEeprom.cpp>
#include <ExternalByteArrayEeprom.h>
#include <ExternalByteArrayEeprom.cpp>
```

Include dependency graph for main.cpp:



Macros

- `#define VIRTUAL_ENVIROMENT 1`

Functions

- `char * itob (int i)`
- `void _rs_io_memory_dump (rs_t *rs)`
- `void resource_dump (rs_resource_t *resource)`
- `void wrapper_format (rs_t *rs)`
- `int main ()`

5.3.1 Macro Definition Documentation

5.3.1.1 `#define VIRTUAL_ENVIROMENT 1`

Definition at line 1 of file [main.cpp](#).

5.3.2 Function Documentation

5.3.2.1 `void _rs_io_memory_dump (rs_t * rs)`

Definition at line 59 of file [main.cpp](#).

5.3.2.2 `char* itob (int i)`

Definition at line 45 of file [main.cpp](#).

5.3.2.3 `int main ()`

Definition at line 173 of file [main.cpp](#).

5.3.2.4 `void resource_dump (rs_resource_t * resource)`

Definition at line 135 of file [main.cpp](#).

5.3.2.5 `void wrapper_format (rs_t * rs)`

Definition at line 148 of file [main.cpp](#).

5.4 main.cpp

```
00001 #define VIRTUAL_ENVIROMENT 1
00002
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <stdint.h>
00006
00007 #include <rs.h>
00008 #include <rs_util.h>
00009 #include <rs_io.c>
00010 #include <rs_util.c>
00011 #include <rs.c>
00012 #include <rs_init_partition.c>
00013
00014 #include <Seekable.h>
00015 #include <Seekable.cpp>
00016 #include <Closeable.h>
00017 #include <Closeable.cpp>
00018 #include <InputStream.h>
00019 #include <InputStream.cpp>
00020 #include <ResourceInputStream.h>
00021 #include <ResourceInputStream.cpp>
00022
00023 #include <OutputStream.h>
00024 #include <OutputStream.cpp>
00025 #include <ResourceOutputStream.h>
00026 #include <ResourceOutputStream.cpp>
00027
00028 #include <SimpleResource.h>
00029 #include <SimpleResourceSystem.h>
00030 #include <SimpleResource.cpp>
00031 #include <SimpleResourceSystem.cpp>
00032 #include <SimpleResourceIO.h>
00033 #include <SimpleResourceIO.cpp>
00034 #include <SimpleVirtualResourceIO.h>
00035 #include <SimpleVirtualResourceIO.cpp>
00036 #include <SimpleExternalEepromResourceIO.h>
00037 #include <SimpleExternalEepromResourceIO.cpp>
00038 #include <SimpleArrayResourceIO.h>
00039 #include <SimpleArrayResourceIO.cpp>
00040 #include <ExternalEeprom.h>
00041 #include <ExternalEeprom.cpp>
00042 #include <ExternalByteArrayEeprom.h>
```

```

00043 #include <ExternalByteArrayEeprom.cpp>
00044
00045 char* itob(int i) {
00046     int bits;
00047     int j, k;
00048     uint16_t mi = 0;
00049     mi |= i;
00050     static char buff[sizeof (mi) * 8 + 1];
00051     bits = sizeof (mi) * 8;
00052     for (j = bits - 1, k = 0; j >= 0; j--, k++) {
00053         buff[k] = ((mi >> j) & 0x01) + '0';
00054     }
00055     buff[bits] = '\0';
00056     return buff;
00057 }
00058
00059 void _rs_io_memory_dump(rs_t *rs) {
00060     rs_memory_address_t memory_address;
00061     uint16_t count, count2;
00062     uint8_t d = 0;
00063     FILE *fp;
00064     if (!_rs_is_driver_monted(rs->driver)) {
00065         printf("Rs not mouted yet\n");
00066         return;
00067     }
00068     fp = fopen("dump", "w+");
00069     fprintf(fp, "DRIVER: %x\n", rs->driver);
00070     fprintf(fp, "\n=====n");
00071     fprintf(fp, "\nRs\n");
00072     fprintf(fp, "-----n");
00073     fprintf(fp, "memory_size:                0x%04x %4d %s\n", rs->
memory_size, rs->memory_size, itob(rs->memory_size));
00074     fprintf(fp, "resource_descriptor_table_address: 0x%04x %4d %s\n", rs->
resource_descriptor_table_address, rs->
resource_descriptor_table_address, itob(rs->
resource_descriptor_table_address));
00075     fprintf(fp, "cluster_table_address:            0x%04x %4d %s\n", rs->
cluster_table_address, rs->cluster_table_address,
itob(rs->cluster_table_address));
00076     fprintf(fp, "sizeof_resource_descriptor_table: 0x%04x %4d %s\n", rs->
sizeof_resource_descriptor_table, rs->
sizeof_resource_descriptor_table, itob(rs->
sizeof_resource_descriptor_table));
00077     fprintf(fp, "sizeof_cluster_table:                0x%04x %4d %s\n", rs->
sizeof_cluster_table, rs->sizeof_cluster_table,
itob(rs->sizeof_resource_descriptor_table));
00078     fprintf(fp, "sizeof_resource_descriptor:          0x%04x %4d %s\n", rs->
sizeof_resource_descriptor, rs->
sizeof_resource_descriptor, itob(rs->
sizeof_resource_descriptor));
00079     fprintf(fp, "sizeof_cluster:                    0x%04x %4d %s\n", rs->
sizeof_cluster, rs->sizeof_cluster, itob(rs->
sizeof_cluster));
00080     fprintf(fp, "resource_descriptor_count:          0x%04x %4d %s\n", rs->
resource_descriptor_count, rs->
resource_descriptor_count, itob(rs->
resource_descriptor_count));
00081     fprintf(fp, "cluster_count:                      0x%04x %4d %s\n", rs->
cluster_count, rs->cluster_count, itob(rs->
cluster_count));
00082     fprintf(fp, "sizeof_cluster_data:                0x%04x %4d %s\n", rs->
sizeof_cluster_data, rs->sizeof_cluster_data,
itob(rs->sizeof_cluster_data));
00083     fprintf(fp, "sizeof_cluster_control:            0x%04x %4d %s\n", rs->
sizeof_cluster_control, rs->sizeof_cluster_control,
itob(rs->sizeof_cluster_control));
00084     fprintf(fp, "free_clusters:                    0x%04x %4d %s\n", rs->
free_clusters, rs->free_clusters, itob(rs->
free_clusters));
00085     fprintf(fp, "flags:                            0x%04x %4d %s\n", rs->flags, rs->
flags, itob(rs->flags));
00086     fprintf(fp, "\n=====n");
00087     fprintf(fp, "\nResource table\n");
00088     fprintf(fp, "-----n");
00089     count = 0;
00090     for (
memory_address = rs->resource_descriptor_table_address;
memory_address < (rs->resource_descriptor_table_address + rs->
sizeof_resource_descriptor_table);
memory_address++)
00091     {
00092
00093     if ((count % rs->sizeof_resource_descriptor) == 0) {
00094         fprintf(fp, "\n%02x: ", (count) ? count / rs->
sizeof_resource_descriptor : 0);
00095     }
00096     fprintf(fp, "%02x ", _rs_io_read(rs->driver, memory_address));

```

```

00100         count++;
00101     }
00102     fprintf(fp, "\n=====\\n");
00103     fprintf(fp, "\\nCluster table\\n");
00104     fprintf(fp, "-----\\n");
00105     fprintf(fp, "\\n      |nn |pp |");
00106     for (count = 0; count < rs->sizeof_cluster_data; count++) {
00107         fprintf(fp, "dd ");
00108     }
00109     fprintf(fp, "\\n      -----");
00110     for (count = 0; count < rs->sizeof_cluster_data; count++) {
00111         fprintf(fp, "----");
00112     }
00113     count = 0;
00114
00115     for (
00116         memory_address = rs->cluster_table_address;
00117         memory_address < (rs->cluster_table_address + rs->
sizeof_cluster_table);
00118         memory_address++
00119     ) {
00120         if ((count % rs->sizeof_cluster) == 0) {
00121             fprintf(fp, "\\n%02x: |", (count) ? count / rs->sizeof_cluster : 0);
00122             count2 = 0;
00123         }
00124         if (count2 == 1 || count2 == 2) {
00125             fprintf(fp, "|");
00126         }
00127         fprintf(fp, "%02x ", (d = _rs_io_read(rs->driver, memory_address)));
00128         fflush(fp);
00129         count++;
00130         count2++;
00131     }
00132     fclose(fp);
00133 }
00134
00135 void resource_dump(rs_resource_t *resource) {
00136     printf("===== resource dump begin =====\\n");
00137     printf(" resource descriptor: %d %s\\n", resource->resource_descriptor,
itob(resource->resource_descriptor));
00138     printf(" first cluster:_____ %d %s\\n", resource->first_cluster,
itob(resource->first_cluster));
00139     printf(" current cluster:_____ %d %s\\n", resource->current_cluster,
itob(resource->current_cluster));
00140     printf(" cluster offset:_____ %d %s\\n", resource->cluster_offset,
itob(resource->cluster_offset));
00141     printf(" size:_____ %d %s\\n", resource->size, itob(resource->
size));
00142     printf(" current position:_____ %d %s\\n", resource->current_position,
itob(resource->current_position));
00143     printf(" flags:_____ %d %s\\n", resource->flags, itob(resource->
flags));
00144     printf(" errors:_____ %d %s\\n", rs_error(resource), itob(
rs_error(resource)));
00145     printf("===== resource dump end =====\\n");
00146 }
00147
00148 void wrapper_format(rs_t *rs) {
00149     uint8_t b[5] = {0xf0, 0x01, 0xff, 0xdd, 0xfa};
00150     uint8_t a[32768];
00151     uint8_t c[32768];
00152     rs->driver = RS_DRIVER_VIRTUAL;
00153     printf("format: %x\\n", SimpleResourceSystem::format(rs));
00154     SimpleResourceSystem rsw(RS_DRIVER_VIRTUAL);
00155     printf("mount: %x\\n", rsw.mount(ResourceSystem::MOUNT_READ_WRITE));
;
00156     SimpleResource rw = rsw.alloc();
00157     printf("code: %x\\n", rw.getCode());
00158     printf("open: %d\\n", rw.open(Resource::OPEN_READ_WRITE));
00159     rw.writeBytes(b, 5);
00160     rw.rewind();
00161     printf("read: %x\\n", rw.read());
00162     printf("size: %x\\n", rw.size());
00163     rw.seek(Resource::SEEK_ORIGIN_BEGIN, 2);
00164     printf("read: %x\\n", rw.read());
00165     printf("availableSpace: %x\\n", rsw.availableSpace());
00166     rw.release();
00167     printf("availableSpace: %x\\n", rsw.availableSpace());
00168     printf("close: %x\\n", rw.close());
00169     printf("open: %d\\n", rw.open(Resource::OPEN_READ_WRITE));
00170     printf("lor: %d\\n", rw.getLastOperationResult());
00171 }
00172
00173 int main() {
00174     rs_t rs;
00175     char *s = (char *) "/tmp/img.bin";
00176     SimpleVirtualResourceIO io(s);

```

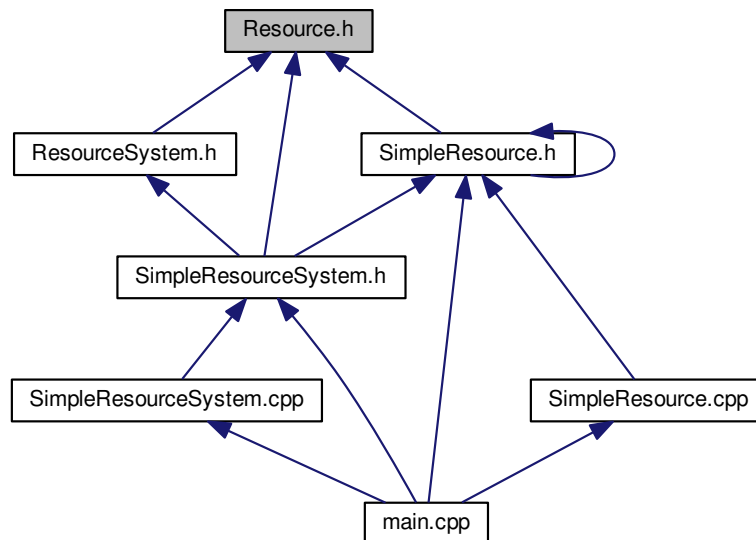
```

00177
00178     printf("Running wrapper specs...\n");
00179
00180     rs_init_partition(&rs, RS_DISK_32K,
00181                     RS_ENV_VIRTUAL);
00181     SimpleResourceIO::associateIO(&io,
00182                                   RS_DRIVER_VIRTUAL);
00182
00183     //wrapper_format(&rs);
00184
00185     printf("Format: %d\n", SimpleResourceSystem::format(&rs));
00186     SimpleResourceSystem srs(RS_DRIVER_VIRTUAL);
00187     printf("Mount: %d\n", srs.mount(ResourceSystem::MOUNT_READ_WRITE));
00188
00188     SimpleResource rw = srs.alloc();
00189     printf("code: %x\n", rw.getCode());
00190     rw.open(Resource::OPEN_READ_WRITE);
00191
00192     ResourceOutputStream ros(&rw);
00193     for (int i = 0; i < 250; i++) {
00194         ros.write(i);
00195     }
00196     printf("rw size: %d\n", rw.size());
00197     ros.close();
00198
00199     rw.open(Resource::OPEN_READ_ONLY);
00200     ResourceInputStream ris(&rw);
00201     while (ris.available()) {
00202         ris.read();
00203     }
00204
00205     rw.close();
00206     _rs_io_memory_dump(srs.getRs());
00207     return 0;
00208 }

```

5.5 Resource.h File Reference

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



Classes

- class [Resource](#)

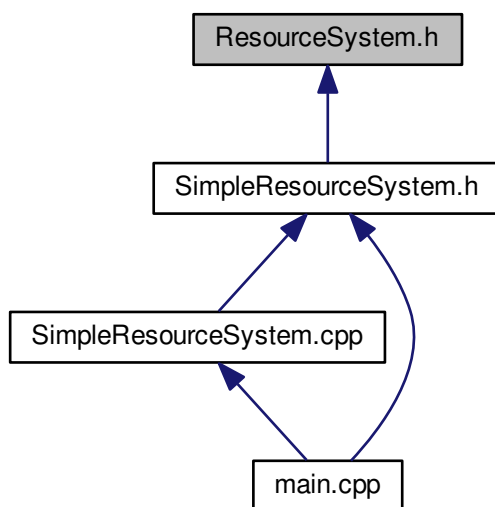
5.6 Resource.h

```

00001
00011 #ifndef __ARDUINO_RESOURCE_H__
00012 #define __ARDUINO_RESOURCE_H__ 1
00013
00014 class Resource {
00015 public:
00016
00017     enum ResourceOperationResult {
00018         OPERATION_SUCCESS = 0,
00019         OPERATION_ERROR_RESOURCE_OPENED = 1,
00020         OPERATION_ERROR_RESOURCE_CLOSED = 2,
00021         OPERATION_ERROR_RESOURCE_READ_ONLY = 3,
00022         OPERATION_ERROR_NO_SPACE_AVAILABLE = 4,
00023         OPERATION_ERROR_DRIVER_BUSY = 5,
00024         OPERATION_ERROR_SEEK_OUT_OF_BOUND = 6,
00025         OPERATION_ERROR_RESOURCE_DOES_NOT_ALLOCATED = 7,
00026         OPERATION_ERROR_DRIVER_NOT_MOUNTED = 8
00027     };
00028
00029     enum OpenOptions {
00030         OPEN_READ_WRITE = 0,
00031         OPEN_READ_ONLY = 1
00032     };
00033
00034     enum ResourceSeekOrigin {
00035         SEEK_ORIGIN_BEGIN = 0,
00036         SEEK_ORIGIN_CURRENT = 1
00037     };
00038
00039     virtual bool open(OpenOptions options) = 0;
00040
00041     virtual bool close() = 0;
00042
00043     virtual void write(unsigned char b) = 0;
00044
00045     virtual void writeBytes(unsigned char* buf, int len) = 0;
00046
00047     virtual int read() = 0;
00048
00049     virtual int readBytes(unsigned char* buf, int len) = 0;
00050
00051     virtual bool seek(ResourceSeekOrigin origin, unsigned int offset) = 0;
00052
00053     virtual bool truncate() = 0;
00054
00055     virtual void sync() = 0;
00056
00057     virtual bool rewind() = 0;
00058
00059     virtual void release() = 0;
00060
00061     virtual unsigned int size() = 0;
00062
00063     virtual unsigned int tell() = 0;
00064
00065     virtual bool eor() = 0;
00066
00067     virtual bool error() = 0;
00068
00069     virtual bool isReadOnly() = 0;
00070 };
00071
00072 #endif // __ARDUINO_RESOURCE_H__

```


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



Classes

- class [ResourceSystem](#)

5.10 ResourceSystem.h

```

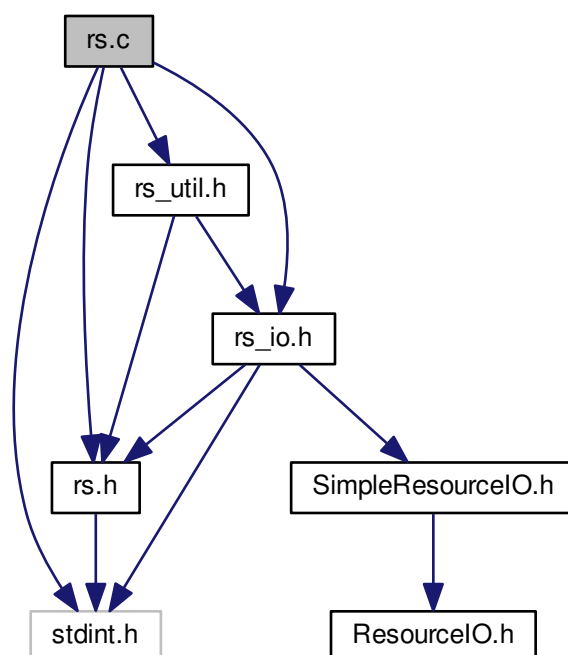
00001
00011 #ifndef __ARDUINO_RESOURCE_SYSTEM_H__
00012 #define __ARDUINO_RESOURCE_SYSTEM_H__ 1
00013
00014 #include <Resource.h>
00015
00016 class ResourceSystem {
00017 public:
00018
00019     enum MountOptions {
00020         MOUNT_READ_WRITE = 0,
00021         MOUNT_READ_ONLY = 1
00022     };
00023
00024     virtual bool mount(MountOptions options) = 0;
00025
00026     virtual bool umount() = 0;
00027
00028     virtual unsigned int totalSpace() = 0;
00029
00030     virtual unsigned int availableSpace() = 0;
00031 };
00032
00033 #endif /* __ARDUINO_RESOURCE_SYSTEM_H__ */
  
```

5.11 rs.c File Reference

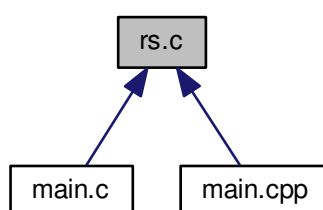
```

#include "rs.h"
#include "rs_io.h"
#include "rs_util.h"
#include <stdint.h>
  
```

Include dependency graph for rs.c:



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



Macros

- `#define __SDCC_RS_C__ 1`

Functions

- `rs_op_result_t rs_format(rs_t *rs)`
- `rs_op_result_t rs_mount(rs_driver_t driver, rs_t *rs, rs_mount_options_t options)`

- [rs_op_result_t rs_umount \(rs_t *rs\)](#)
- [rs_op_result_t rs_open \(rs_t *rs, rs_resource_code_t resource_code, rs_resource_t *resource, rs_open_options_t options\)](#)
- [rs_op_result_t rs_close \(rs_t *rs, rs_resource_t *resource\)](#)
- [uint8_t rs_read \(rs_t *rs, rs_resource_t *resource\)](#)
- [rs_op_result_t rs_write \(rs_t *rs, rs_resource_t *resource, uint8_t data_to_write\)](#)
- [rs_op_result_t rs_seek \(rs_t *rs, rs_resource_t *resource, rs_seek_origin_t origin, rs_seek_int_t offset\)](#)
- [rs_op_result_t rs_truncate \(rs_t *rs, rs_resource_t *resource\)](#)
- [void rs_sync \(rs_t *rs, rs_resource_t *resource\)](#)
- [void rs_stat \(rs_t *rs, rs_resource_t *resource, rs_stat_t *stat\)](#)
- [rs_op_result_t rs_rewind \(rs_t *rs, rs_resource_t *resource\)](#)
- [rs_resource_code_t rs_alloc \(rs_t *rs\)](#)
- [uint8_t rs_release \(rs_t *rs, rs_resource_t *resource\)](#)
- [rs_resource_size_t rs_size \(rs_resource_t *resource\)](#)
- [rs_resource_size_t rs_tell \(rs_resource_t *resource\)](#)
- [uint8_t rs_eor \(rs_resource_t *resource\)](#)
- [uint8_t rs_error \(rs_resource_t *resource\)](#)
- [rs_resource_size_t rs_available_space \(rs_t *rs\)](#)
- [rs_resource_size_t rs_total_space \(rs_t *rs\)](#)

Variables

- [rs_global_flags_t rs_global_flags](#)

5.11.1 Macro Definition Documentation

5.11.1.1 [#define __SDCC_RS_C__ 1](#)

SDCC - PIC resource system.

[rs.c](#)

A file system implementation based on the idea of resources

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line [12](#) of file [rs.c](#).

5.11.2 Function Documentation

5.11.2.1 [rs_resource_code_t rs_alloc \(rs_t * rs \)](#)

Definition at line [223](#) of file [rs.c](#).

5.11.2.2 [rs_resource_size_t rs_available_space \(rs_t * rs \)](#)

Definition at line [281](#) of file [rs.c](#).

5.11.2.3 [rs_op_result_t rs_close \(rs_t * rs, rs_resource_t * resource \)](#)

Definition at line [90](#) of file [rs.c](#).

5.11.2.4 [uint8_t rs_eor \(rs_resource_t * resource \)](#)

Definition at line [273](#) of file [rs.c](#).

5.11.2.5 `uint8_t rs_error (rs_resource_t * resource)`

Definition at line 277 of file [rs.c](#).

5.11.2.6 `rs_op_result_t rs_format (rs_t * rs)`

Definition at line 21 of file [rs.c](#).

5.11.2.7 `rs_op_result_t rs_mount (rs_driver_t driver, rs_t * rs, rs_mount_options_t options)`

Definition at line 33 of file [rs.c](#).

5.11.2.8 `rs_op_result_t rs_open (rs_t * rs, rs_resource_code_t resource_code, rs_resource_t * resource, rs_open_resource_options_t options)`

Definition at line 54 of file [rs.c](#).

5.11.2.9 `uint8_t rs_read (rs_t * rs, rs_resource_t * resource)`

Definition at line 97 of file [rs.c](#).

5.11.2.10 `uint8_t rs_release (rs_t * rs, rs_resource_t * resource)`

Definition at line 251 of file [rs.c](#).

5.11.2.11 `rs_op_result_t rs_rewind (rs_t * rs, rs_resource_t * resource)`

Definition at line 215 of file [rs.c](#).

5.11.2.12 `rs_op_result_t rs_seek (rs_t * rs, rs_resource_t * resource, rs_seek_origin_t origin, rs_seek_int_t offset)`

Definition at line 148 of file [rs.c](#).

5.11.2.13 `rs_resource_size_t rs_size (rs_resource_t * resource)`

Definition at line 265 of file [rs.c](#).

5.11.2.14 `void rs_stat (rs_t * rs, rs_resource_t * resource, rs_stat_t * stat)`

Definition at line 211 of file [rs.c](#).

5.11.2.15 `void rs_sync (rs_t * rs, rs_resource_t * resource)`

Definition at line 202 of file [rs.c](#).

5.11.2.16 `rs_resource_size_t rs_tell (rs_resource_t * resource)`

Definition at line 269 of file [rs.c](#).

5.11.2.17 `rs_resource_size_t rs_total_space (rs_t * rs)`

Definition at line 285 of file [rs.c](#).

5.11.2.18 `rs_op_result_t rs_truncate (rs_t * rs, rs_resource_t * resource)`

Definition at line 183 of file [rs.c](#).

5.11.2.19 `rs_op_result_t rs_umount (rs_t * rs)`

Definition at line 47 of file [rs.c](#).

5.11.2.20 rs_op_result_t rs_write (rs_t* rs, rs_resource_t* resource, uint8_t data_to_write)

Definition at line 116 of file [rs.c](#).

5.11.3 Variable Documentation

5.11.3.1 rs_global_flags_t rs_global_flags

Definition at line 19 of file [rs.c](#).

5.12 rs.c

```

00001
00011 #ifndef __SDCC_RS_C__
00012 #define __SDCC_RS_C__ 1
00013
00014 #include "rs.h"
00015 #include "rs_io.h"
00016 #include "rs_util.h"
00017 #include <stdint.h>
00018
00019 rs_global_flags_t rs_global_flags;
00020
00021 rs_op_result_t rs_format(rs_t *rs) {
00022     uint8_t i;
00023     _rs_write_rs_to_disc(rs->driver, rs);
00024     for (i = 0; i < rs->resource_descriptor_count; i++) {
00025         _rs_format_resource_descriptor(rs, i);
00026     }
00027     for (i = 0; i < rs->cluster_count; i++) {
00028         _rs_format_cluster(rs, i);
00029     }
00030     return RS_OP_RESULT_SUCCESS;
00031 }
00032
00033 rs_op_result_t rs_mount(rs_driver_t driver,
00034     rs_t *rs, rs_mount_options_t options) {
00035     if (_rs_is_driver_mounted(driver)) {
00036         return RS_OP_RESULT_ERROR_DRIVER_BUSY;
00037     }
00038     _rs_read_rs_from_disc(driver, rs);
00039     _rs_set_driver_mounted(driver, 1);
00040     if (options & RS_MOUNT_OPTION_READ_ONLY) {
00041         rs->flags |= RS_FLAG_BIT_READ_ONLY;
00042     }
00043     rs->driver = driver;
00044     _rs_free_resource_descriptors(rs);
00045     return RS_OP_RESULT_SUCCESS;
00046 }
00047 rs_op_result_t rs_umount(rs_t *rs) {
00048     if (_rs_is_driver_mounted(rs->driver)) {
00049         _rs_set_driver_mounted(rs->driver, 0);
00050     }
00051     return RS_OP_RESULT_SUCCESS;
00052 }
00053
00054 rs_op_result_t rs_open(rs_t *rs, rs_resource_code_t
00055     resource_code, rs_resource_t *resource, rs_open_resource_options_t options) {
00056     uint8_t i;
00057     rs_memory_address_t address;
00058     rs_resource_descriptor_t resource_descriptor;
00059     uint8_t flags;
00060     if (!_rs_is_driver_mounted(rs->driver)) {
00061         return RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED;
00062     }
00063     resource_descriptor = _rs_resource_code_to_resource_descriptor(
00064         resource_code);
00065     address = _rs_resource_descriptor_to_address(rs, resource_descriptor);
00066
00067     flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(address));
00068     if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00069         return RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED;
00070     }
00071     if (flags & RS_RESOURCE_FLAG_BIT_OPENED) {
00072         return RS_OP_RESULT_ERROR_RESOURCE_OPENED;
00073     }
00074     flags |= RS_RESOURCE_FLAG_BIT_OPENED;

```

```

00072     if ((options & RS_OPEN_RESOURCE_OPTION_READ_ONLY) || (rs->
00073         flags & RS_FLAG_BIT_READ_ONLY)) {
00074         flags |= RS_RESOURCE_FLAG_BIT_READ_ONLY;
00075     }
00076     _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(address), flags);
00077     resource->resource_descriptor = resource_descriptor;
00078     resource->first_cluster = _rs_io_read(rs->driver,
00079         RD_ADDRESS_TO_FIRST_CLUSTER(address));
00079     resource->current_cluster = resource->first_cluster;
00080     resource->cluster_offset = rs->sizeof_cluster_control;
00081     resource->current_position = 0;
00082     for (i = 0; i < RS_SIZEOF_RESOURCE_SIZE; i++) {
00083         *((uint8_t *) (&resource->size) + i) = _rs_io_read(rs->
00084             driver, address + i);
00085     }
00085     resource->flags = flags;
00086     _rs_check_for_eor_reached(resource);
00087     return RS_OP_RESULT_SUCCESS;
00088 }
00089
00090 rs_op_result_t rs_close(rs_t *rs, rs_resource_t *resource) {
00091     rs_sync(rs, resource);
00092     _rs_free_resource_descriptor(rs, resource->
00093         resource_descriptor);
00093     resource->flags = ~RS_RESOURCE_FLAG_BIT_OPENED;
00094     return RS_OP_RESULT_SUCCESS;
00095 }
00096
00097 uint8_t rs_read(rs_t *rs, rs_resource_t *resource) {
00098     rs_memory_address_t address;
00099     uint8_t read_data;
00100     if (!(resource->flags & RS_RESOURCE_FLAG_BIT_OPENED)) {
00101         resource->flags |= RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
00102     ;
00102         return 0;
00103     }
00104     if (_rs_is_eor_reached(resource)) {
00105         return 0;
00106     }
00107     _rs_check_for_availability(rs, resource);
00108     address = _rs_cluster_to_address(rs, resource->
00109         current_cluster);
00109     read_data = _rs_io_read(rs->driver, address + resource->
00110         cluster_offset);
00110     resource->current_position++;
00111     resource->cluster_offset++;
00112     _rs_check_for_eor_reached(resource);
00113     return read_data;
00114 }
00115
00116 rs_op_result_t rs_write(rs_t *rs, rs_resource_t *resource, uint8_t
00117     data_to_write) {
00117     rs_memory_address_t address;
00118     if (!(resource->flags & RS_RESOURCE_FLAG_BIT_OPENED)) {
00119         return RS_OP_RESULT_ERROR_RESOURCE_CLOSED;
00120     }
00121     if (resource->flags & RS_RESOURCE_FLAG_BIT_READ_ONLY) {
00122         return RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY;
00123     }
00124     if (!_rs_check_for_availability(rs, resource)) {
00125         return RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE;
00126     }
00127     address = _rs_cluster_to_address(rs, resource->
00128         current_cluster);
00128     _rs_io_write(rs->driver, address + resource->
00129         cluster_offset, data_to_write);
00129     resource->cluster_offset++;
00130     resource->current_position++;
00131     if (rs_eor(resource)) {
00132         resource->size++;
00133     }
00134     /*
00135     * It is causing seriously performance problems. Since the IO has a
00136     * buffer, and we are writing at this buffer, once we need to sync, this
00137     * buffer will be flushed to the resource and the IO pointer will need
00138     * to go to the resource descriptor to write the new size of the
00139     * resource. After the sync, we will continue to write the next byte,
00140     * the buffer filled on the sync will be lost and a new buffer will be
00141     * created to write the next byte... and so on.
00142     */
00143     //rs_sync(rs, resource);
00144 }
00145     return RS_OP_RESULT_SUCCESS;
00146 }
00147
00148 rs_op_result_t rs_seek(rs_t *rs, rs_resource_t *resource,

```

```

    rs_seek_origin_t origin, rs_seek_int_t offset) {
00149     int16_t new_position = 0;
00150     if (resource->size == 0) {
00151         return RS_OP_RESULT_SUCCESS;
00152     }
00153     switch (origin) {
00154         case RS_SEEK_ORIGIN_BEGIN:
00155             new_position = offset;
00156             break;
00157         case RS_SEEK_ORIGIN_CURRENT:
00158             new_position = resource->current_position + offset;
00159             break;
00160     }
00161     new_position %= resource->size + 1;
00162     if (new_position < 0) {
00163         new_position += resource->size;
00164     }
00165     if (new_position == 0) {
00166         rs_rewind(rs, resource);
00167         return RS_OP_RESULT_SUCCESS;
00168     }
00169     if (new_position < resource->current_position) {
00170         if (new_position > (resource->current_position - new_position)) {
00171             _rs_move_current_position_back(rs, resource, (resource->
current_position - new_position));
00172         } else {
00173             rs_rewind(rs, resource);
00174             _rs_move_current_position_ahead(rs, resource, new_position);
00175         }
00176     } else {
00177         _rs_move_current_position_ahead(rs, resource, (new_position -
resource->current_position));
00178     }
00179     _rs_check_for_eor_reached(resource);
00180     return RS_OP_RESULT_SUCCESS;
00181 }
00182
00183 rs_op_result_t rs_truncate(rs_t *rs, rs_resource_t *resource) {
00184     uint8_t flags;
00185     rs_memory_address_t resource_descriptor_address;
00186     uint8_t freed_clusters = 0;
00187     resource_descriptor_address = _rs_resource_descriptor_to_address(rs,
resource->resource_descriptor);
00188     flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(
resource_descriptor_address));
00189     if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00190         return RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED
;
00191     }
00192     if (resource->size > rs->sizeof_cluster_data) {
00193         freed_clusters = _rs_format_clusters_chain(rs,
_rs_next_cluster_by_cluster(rs, resource->
first_cluster));
00194     }
00195     _rs_increase_free_clusters(rs, freed_clusters);
00196     resource->size = 0x00;
00197     _rs_io_write(rs->driver, RD_ADDRESS_TO_SIZE_LOW(
resource_descriptor_address), 0x00);
00198     _rs_io_write(rs->driver, RD_ADDRESS_TO_SIZE_HIGH(
resource_descriptor_address), 0x00);
00199     return RS_OP_RESULT_SUCCESS;
00200 }
00201
00202 void rs_sync(rs_t *rs, rs_resource_t *resource) {
00203     uint8_t i;
00204     rs_memory_address_t address;
00205     address = _rs_resource_descriptor_to_address(rs, resource->
resource_descriptor);
00206     for (i = 0; i < 2; i++) {
00207         _rs_io_write(rs->driver, address + i, *((uint8_t *) (&(resource->
size)) + i));
00208     }
00209 }
00210
00211 void rs_stat(rs_t *rs, rs_resource_t *resource,
rs_stat_t *stat) { // TODO
00212     stat->flags = 0xff;
00213 }
00214
00215 rs_op_result_t rs_rewind(rs_t *rs, rs_resource_t *resource) {
00216     resource->current_cluster = resource->first_cluster;
00217     resource->cluster_offset = rs->sizeof_cluster_control;
00218     resource->current_position = 0;
00219     _rs_check_for_eor_reached(resource);
00220     return RS_OP_RESULT_SUCCESS;
00221 }
00222

```



```

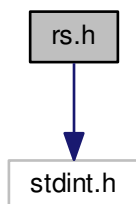
00223 rs_resource_code_t rs_alloc(rs_t *rs) {
00224     uint8_t i;
00225     uint8_t flags;
00226     rs_cluster_t first_cluster;
00227     rs_memory_address_t resource_descriptor_address, cluster_address;
00228     if (rs->free_clusters < 1) {
00229         return RS_NULL_RESOURCE_CODE;
00230     }
00231     resource_descriptor_address = rs->resource_descriptor_table_address;
00232     for (i = 0; i < rs->resource_descriptor_count; i++) {
00233         flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(
resource_descriptor_address));
00234         if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00235             cluster_address = _rs_alloc_cluster(rs);
00236             if (cluster_address == RS_NULL_CLUSTER_ADDRESS) {
00237                 return RS_NULL_RESOURCE_CODE;
00238             }
00239             flags |= RS_RESOURCE_FLAG_BIT_ALLOCATED;
00240             first_cluster = _rs_address_to_cluster(rs, cluster_address);
00241             _rs_create_cluster_chain(rs, first_cluster,
RS_INEXISTENT_CLUSTER);
00242             _rs_io_write(rs->driver,
RD_ADDRESS_TO_FIRST_CLUSTER(resource_descriptor_address), first_cluster);
00243             _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(
resource_descriptor_address), flags);
00244             return _rs_resource_descriptor_to_resource_code(i);
00245         }
00246         resource_descriptor_address += rs->sizeof_resource_descriptor;
00247     }
00248     return RS_NULL_RESOURCE_CODE;
00249 }
00250
00251 uint8_t rs_release(rs_t *rs, rs_resource_t *resource) {
00252     uint8_t flags;
00253     rs_memory_address_t resource_descriptor_address;
00254     resource_descriptor_address = _rs_resource_descriptor_to_address(rs,
resource->resource_descriptor);
00255     flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(
resource_descriptor_address));
00256     if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
00257         return 1;
00258     }
00259     _rs_format_resource_clusters(rs, resource);
00260     _rs_format_resource_descriptor(rs, resource->
resource_descriptor);
00261     resource->flags = 0x00;
00262     return 1;
00263 }
00264
00265 rs_resource_size_t rs_size(rs_resource_t *resource) {
00266     return resource->size;
00267 }
00268
00269 rs_resource_size_t rs_tell(rs_resource_t *resource) {
00270     return resource->current_position;
00271 }
00272
00273 uint8_t rs_eor(rs_resource_t *resource) {
00274     return _rs_is_eor_reached(resource);
00275 }
00276
00277 uint8_t rs_error(rs_resource_t *resource) {
00278     return (resource->flags & RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
|| resource->flags & RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE)
;
00279 }
00280
00281 rs_resource_size_t rs_available_space(rs_t *rs) {
00282     return rs->free_clusters * rs->sizeof_cluster_data;
00283 }
00284
00285 rs_resource_size_t rs_total_space(rs_t *rs) {
00286     return rs->cluster_count * rs->sizeof_cluster_data;
00287 }
00288
00289 #endif // __SDCC_RS_C__

```

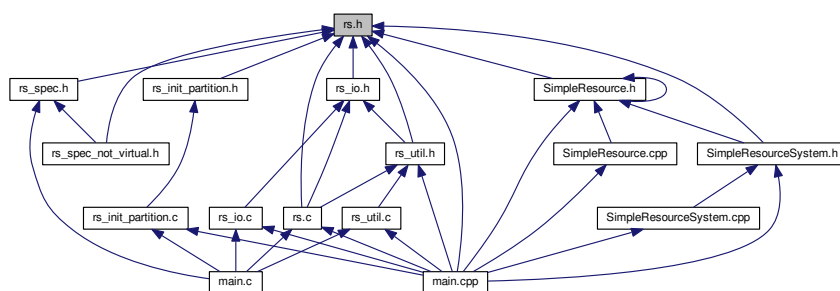
5.13 rs.h File Reference

```
#include <stdint.h>
```

Include dependency graph for rs.h:



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



Classes

- struct [rs_stat_t](#)
- struct [rs_t](#)
- struct [rs_resource_t](#)
- struct [rs_global_flags_t](#)

Macros

- `#define RS_NULL_RESOURCE_CODE 0xff`
- `#define RS_NULL_CLUSTER 0xff`
- `#define RS_NULL_RESOURCE_DESCRIPTOR_ADDRESS 0xff`
- `#define RS_NULL_CLUSTER_ADDRESS 0x00`
- `#define RS_FIRST_ADDRESS_OF_MEMORY 0x00`
- `#define RS_SIZEOF_RESOURCE_SIZE 0x02`
- `#define RS_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 `rs_resource_descriptor_t`
- typedef uint8_t `rs_cluster_t`
- typedef uint16_t `rs_resource_size_t`
- typedef uint16_t `rs_memory_address_t`
- typedef uint8_t `rs_resource_code_t`
- typedef uint16_t `rs_seek_int_t`

Enumerations

- enum `rs_driver_t` {
`RS_DRIVER_VIRTUAL` = 0, `RS_DRIVER_SELF_EEPROM` = 1, `RS_DRIVER_MULTI_EXTERNAL_EEP`←
`ROM` = 2, `RS_DRIVER_EXTERNAL_EEPROM` = 3,
`RS_DRIVER_ARDUINO_EEPROM` = 4 }
- enum `rs_resource_flag_bits_t` {
`RS_RESOURCE_FLAG_BIT_OPENED` = 1, `RS_RESOURCE_FLAG_BIT_READ_ONLY` = 2, `RS_RESO`←
`URCE_FLAG_BIT_ERROR_ON_LAST_READ` = 4, `RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_W`←
`RITE` = 8,
`RS_RESOURCE_FLAG_BIT_ALLOCATED` = 16, `RS_RESOURCE_FLAG_BIT_EOR_REACHED` = 32 }
- enum `rs_open_resource_options_t` { `RS_OPEN_RESOURCE_OPTION_NORMAL` = 0, `RS_OPEN_RESO`←
`URCE_OPTION_READ_ONLY` = 1 }
- enum `rs_mount_options_t` { `RS_MOUNT_OPTION_NORMAL` = 0, `RS_MOUNT_OPTION_READ_ONLY` = 1
}
- enum `rs_flag_bits_t` { `RS_FLAG_BIT_DRIVER_MOUNTED` = 1, `RS_FLAG_BIT_READ_ONLY` = 2 }
- enum `rs_op_result_t` {
`RS_OP_RESULT_SUCCESS` = 0, `RS_OP_RESULT_ERROR_RESOURCE_OPENED` = 1, `RS_OP_RES`←
`ULT_ERROR_RESOURCE_CLOSED` = 2, `RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY` = 3,
`RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE` = 4, `RS_OP_RESULT_ERROR_DRIVER_BUSY` = 5,
`RS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND` = 6, `RS_OP_RESULT_ERROR_RESOURCE_DO`←
`ES_NOT_ALLOCATED` = 7,
`RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED` = 8 }
- enum `rs_seek_origin_t` { `RS_SEEK_ORIGIN_BEGIN` = 0, `RS_SEEK_ORIGIN_CURRENT` = 1 }

Functions

- `rs_op_result_t rs_format` (`rs_t` *rs)
- `rs_op_result_t rs_mount` (`rs_driver_t` driver, `rs_t` *rs, `rs_mount_options_t` options)
- `rs_op_result_t rs_umount` (`rs_t` *rs)
- `rs_op_result_t rs_open` (`rs_t` *rs, `rs_resource_code_t` resource_code, `rs_resource_t` *resource, `rs_open_`←
`resource_options_t` options)
- `rs_op_result_t rs_close` (`rs_t` *rs, `rs_resource_t` *resource)
- `uint8_t rs_read` (`rs_t` *rs, `rs_resource_t` *resource)
- `rs_op_result_t rs_write` (`rs_t` *rs, `rs_resource_t` *resource, uint8_t data_to_write)
- `rs_op_result_t rs_seek` (`rs_t` *rs, `rs_resource_t` *resource, `rs_seek_origin_t` origin, `rs_seek_int_t` offset)
- `rs_op_result_t rs_truncate` (`rs_t` *rs, `rs_resource_t` *resource)
- void `rs_sync` (`rs_t` *rs, `rs_resource_t` *resource)
- void `rs_stat` (`rs_t` *rs, `rs_resource_t` *resource, `rs_stat_t` *stat)
- `rs_op_result_t rs_rewind` (`rs_t` *rs, `rs_resource_t` *resource)
- `rs_resource_code_t rs_alloc` (`rs_t` *rs)
- `uint8_t rs_release` (`rs_t` *rs, `rs_resource_t` *resource)

- [rs_resource_size_t rs_size \(rs_resource_t *resource\)](#)
- [rs_resource_size_t rs_tell \(rs_resource_t *resource\)](#)
- [uint8_t rs_eor \(rs_resource_t *resource\)](#)
- [uint8_t rs_error \(rs_resource_t *resource\)](#)
- [rs_resource_size_t rs_available_space \(rs_t *rs\)](#)
- [rs_resource_size_t rs_total_space \(rs_t *rs\)](#)

Variables

- [rs_global_flags_t rs_global_flags](#)

5.13.1 Macro Definition Documentation

5.13.1.1 `#define CLUSTER_ADDRESS_TO_DATA(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 2)`

Definition at line 30 of file [rs.h](#).

5.13.1.2 `#define CLUSTER_ADDRESS_TO_NEXT(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 0)`

Definition at line 28 of file [rs.h](#).

5.13.1.3 `#define CLUSTER_ADDRESS_TO_PREV(CLUSTER_ADDRESS) ((CLUSTER_ADDRESS) + 1)`

Definition at line 29 of file [rs.h](#).

5.13.1.4 `#define RD_ADDRESS_TO_FIRST_CLUSTER(RD_ADDRESS) ((RD_ADDRESS) + 2)`

Definition at line 34 of file [rs.h](#).

5.13.1.5 `#define RD_ADDRESS_TO_FLAG(RD_ADDRESS) ((RD_ADDRESS) + 3)`

Definition at line 35 of file [rs.h](#).

5.13.1.6 `#define RD_ADDRESS_TO_SIZE_HIGH(RD_ADDRESS) ((RD_ADDRESS) + 1)`

Definition at line 33 of file [rs.h](#).

5.13.1.7 `#define RD_ADDRESS_TO_SIZE_LOW(RD_ADDRESS) ((RD_ADDRESS) + 0)`

Definition at line 32 of file [rs.h](#).

5.13.1.8 `#define RS_FIRST_ADDRESS_OF_MEMORY 0x00`

Definition at line 22 of file [rs.h](#).

5.13.1.9 `#define RS_INEXISTENT_CLUSTER 0xff`

Definition at line 26 of file [rs.h](#).

5.13.1.10 `#define RS_NULL_CLUSTER 0xff`

Definition at line 17 of file [rs.h](#).

5.13.1.11 `#define RS_NULL_CLUSTER_ADDRESS 0x00`

Definition at line 20 of file [rs.h](#).

5.13.1.12 `#define RS_NULL_RESOURCE_DESCRIPTOR_ADDRESS 0xff`

Definition at line 19 of file [rs.h](#).

5.13.1.13 #define RS_NULL_RESOURCE_CODE 0xff

SDCC - PIC resource system.

[rs.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 [rs.h](#).

5.13.1.14 #define RS_SIZEOF_RESOURCE_SIZE 0x02

Definition at line 24 of file [rs.h](#).

5.13.2 Typedef Documentation

5.13.2.1 typedef uint8_t rs_cluster_t

Definition at line 38 of file [rs.h](#).

5.13.2.2 typedef uint16_t rs_memory_address_t

Definition at line 40 of file [rs.h](#).

5.13.2.3 typedef uint8_t rs_resource_code_t

Definition at line 41 of file [rs.h](#).

5.13.2.4 typedef uint8_t rs_resource_descriptor_t

Definition at line 37 of file [rs.h](#).

5.13.2.5 typedef uint16_t rs_resource_size_t

Definition at line 39 of file [rs.h](#).

5.13.2.6 typedef uint16_t rs_seek_int_t

Definition at line 42 of file [rs.h](#).

5.13.3 Enumeration Type Documentation

5.13.3.1 enum rs_driver_t

Enumerator

RS_DRIVER_VIRTUAL

RS_DRIVER_SELF_EEPROM

RS_DRIVER_MULTI_EXTERNAL_EEPROM

RS_DRIVER_EXTERNAL_EEPROM

RS_DRIVER_ARDUINO_EEPROM

Definition at line 46 of file [rs.h](#).

5.13.3.2 enum rs_flag_bits_t

Enumerator

RS_FLAG_BIT_DRIVER_MOUNTED
RS_FLAG_BIT_READ_ONLY

Definition at line 81 of file [rs.h](#).

5.13.3.3 enum rs_mount_options_t

Enumerator

RS_MOUNT_OPTION_NORMAL
RS_MOUNT_OPTION_READ_ONLY

Definition at line 74 of file [rs.h](#).

5.13.3.4 enum rs_op_result_t

Enumerator

RS_OP_RESULT_SUCCESS
RS_OP_RESULT_ERROR_RESOURCE_OPENED
RS_OP_RESULT_ERROR_RESOURCE_CLOSED
RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY
RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE
RS_OP_RESULT_ERROR_DRIVER_BUSY
RS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND
RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED
RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED

Definition at line 88 of file [rs.h](#).

5.13.3.5 enum rs_open_resource_options_t

Enumerator

RS_OPEN_RESOURCE_OPTION_NORMAL
RS_OPEN_RESOURCE_OPTION_READ_ONLY

Definition at line 67 of file [rs.h](#).

5.13.3.6 enum rs_resource_flag_bits_t

Enumerator

RS_RESOURCE_FLAG_BIT_OPENED
RS_RESOURCE_FLAG_BIT_READ_ONLY
RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE
RS_RESOURCE_FLAG_BIT_ALLOCATED
RS_RESOURCE_FLAG_BIT_EOR_REACHED

Definition at line 56 of file [rs.h](#).

5.13.3.7 enum rs_seek_origin_t

Enumerator

RS_SEEK_ORIGIN_BEGIN

RS_SEEK_ORIGIN_CURRENT

Definition at line 102 of file [rs.h](#).

5.13.4 Function Documentation

5.13.4.1 rs_resource_code_t rs_alloc (rs_t * rs)

Definition at line 223 of file [rs.c](#).

5.13.4.2 rs_resource_size_t rs_available_space (rs_t * rs)

Definition at line 281 of file [rs.c](#).

5.13.4.3 rs_op_result_t rs_close (rs_t * rs, rs_resource_t * resource)

Definition at line 90 of file [rs.c](#).

5.13.4.4 uint8_t rs_eor (rs_resource_t * resource)

Definition at line 273 of file [rs.c](#).

5.13.4.5 uint8_t rs_error (rs_resource_t * resource)

Definition at line 277 of file [rs.c](#).

5.13.4.6 rs_op_result_t rs_format (rs_t * rs)

Definition at line 21 of file [rs.c](#).

5.13.4.7 rs_op_result_t rs_mount (rs_driver_t driver, rs_t * rs, rs_mount_options_t options)

Definition at line 33 of file [rs.c](#).

5.13.4.8 rs_op_result_t rs_open (rs_t * rs, rs_resource_code_t resource_code, rs_resource_t * resource, rs_open_resource_options_t options)

Definition at line 54 of file [rs.c](#).

5.13.4.9 uint8_t rs_read (rs_t * rs, rs_resource_t * resource)

Definition at line 97 of file [rs.c](#).

5.13.4.10 uint8_t rs_release (rs_t * rs, rs_resource_t * resource)

Definition at line 251 of file [rs.c](#).

5.13.4.11 rs_op_result_t rs_rewind (rs_t * rs, rs_resource_t * resource)

Definition at line 215 of file [rs.c](#).

5.13.4.12 rs_op_result_t rs_seek (rs_t * rs, rs_resource_t * resource, rs_seek_origin_t origin, rs_seek_int_t offset)

Definition at line 148 of file [rs.c](#).

5.13.4.13 `rs_resource_size_t rs_size (rs_resource_t * resource)`

Definition at line 265 of file [rs.c](#).

5.13.4.14 `void rs_stat (rs_t * rs, rs_resource_t * resource, rs_stat_t * stat)`

Definition at line 211 of file [rs.c](#).

5.13.4.15 `void rs_sync (rs_t * rs, rs_resource_t * resource)`

Definition at line 202 of file [rs.c](#).

5.13.4.16 `rs_resource_size_t rs_tell (rs_resource_t * resource)`

Definition at line 269 of file [rs.c](#).

5.13.4.17 `rs_resource_size_t rs_total_space (rs_t * rs)`

Definition at line 285 of file [rs.c](#).

5.13.4.18 `rs_op_result_t rs_truncate (rs_t * rs, rs_resource_t * resource)`

Definition at line 183 of file [rs.c](#).

5.13.4.19 `rs_op_result_t rs_ument (rs_t * rs)`

Definition at line 47 of file [rs.c](#).

5.13.4.20 `rs_op_result_t rs_write (rs_t * rs, rs_resource_t * resource, uint8_t data_to_write)`

Definition at line 116 of file [rs.c](#).

5.13.5 Variable Documentation

5.13.5.1 `rs_global_flags_t rs_global_flags`

Definition at line 19 of file [rs.c](#).

5.14 `rs.h`

```

00001
00011 #ifndef __SDCC_RS_H__
00012 #define __SDCC_RS_H__ 1
00013
00014 #include <stdint.h>
00015
00016 #define RS_NULL_RESOURCE_CODE 0xff
00017 #define RS_NULL_CLUSTER 0xff
00018
00019 #define RS_NULL_RESOURCE_DESCRIPTOR_ADDRESS 0xff
00020 #define RS_NULL_CLUSTER_ADDRESS 0x00
00021
00022 #define RS_FIRST_ADDRESS_OF_MEMORY 0x00
00023
00024 #define RS_SIZEOF_RESOURCE_SIZE 0x02
00025
00026 #define RS_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 rs_resource_descriptor_t;

```



```

00038 typedef uint8_t rs_cluster_t;
00039 typedef uint16_t rs_resource_size_t;
00040 typedef uint16_t rs_memory_address_t;
00041 typedef uint8_t rs_resource_code_t;
00042 typedef uint16_t rs_seek_int_t;
00043
00044 // Drivers
00045
00046 typedef enum {
00047     RS_DRIVER_VIRTUAL = 0,
00048     RS_DRIVER_SELF_EEPROM = 1,
00049     RS_DRIVER_MULTI_EXTERNAL_EEPROM = 2,
00050     RS_DRIVER_EXTERNAL_EEPROM = 3,
00051     RS_DRIVER_ARDUINO_EEPROM = 4
00052 } rs_driver_t;
00053
00054 // Resource fag bit values
00055
00056 typedef enum {
00057     RS_RESOURCE_FLAG_BIT_OPENED = 1,
00058     RS_RESOURCE_FLAG_BIT_READ_ONLY = 2,
00059     RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ = 4,
00060     RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE = 8,
00061     RS_RESOURCE_FLAG_BIT_ALLOCATED = 16,
00062     RS_RESOURCE_FLAG_BIT_EOR_REACHED = 32
00063 } rs_resource_flag_bits_t;
00064
00065 // Options to open a resource
00066
00067 typedef enum {
00068     RS_OPEN_RESOURCE_OPTION_NORMAL = 0,
00069     RS_OPEN_RESOURCE_OPTION_READ_ONLY = 1
00070 } rs_open_resource_options_t;
00071
00072 // Options to mount a resource
00073
00074 typedef enum {
00075     RS_MOUNT_OPTION_NORMAL = 0,
00076     RS_MOUNT_OPTION_READ_ONLY = 1
00077 } rs_mount_options_t;
00078
00079 // Rs fag bit values
00080
00081 typedef enum {
00082     RS_FLAG_BIT_DRIVER_MOUNTED = 1,
00083     RS_FLAG_BIT_READ_ONLY = 2
00084 } rs_flag_bits_t;
00085
00086 // Operation result
00087
00088 typedef enum {
00089     RS_OP_RESULT_SUCCESS = 0,
00090     RS_OP_RESULT_ERROR_RESOURCE_OPENED = 1,
00091     RS_OP_RESULT_ERROR_RESOURCE_CLOSED = 2,
00092     RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY = 3,
00093     RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE = 4,
00094     RS_OP_RESULT_ERROR_DRIVER_BUSY = 5,
00095     RS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND = 6,
00096     RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED = 7,
00097     RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED = 8
00098 } rs_op_result_t;
00099
00100 // Seek position reference
00101
00102 typedef enum {
00103     RS_SEEK_ORIGIN_BEGIN = 0,
00104     RS_SEEK_ORIGIN_CURRENT = 1
00105 } rs_seek_origin_t;
00106
00107 typedef struct {
00108     uint8_t flags;
00109 } rs_stat_t;
00110
00111 // Resource system
00112
00113 typedef struct {
00114     rs_driver_t driver;
00115     uint16_t memory_size;
00116     rs_memory_address_t resource_descriptor_table_address
00117 ;
00118     rs_memory_address_t cluster_table_address;
00119     uint16_t sizeof_resource_descriptor_table;
00120     uint16_t sizeof_cluster_table;
00121     uint8_t sizeof_resource_descriptor;
00122     uint8_t sizeof_cluster;
00123     uint8_t resource_descriptor_count;
00124     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 } rs_t;
00129
00130 // Resource
00131
00132 typedef struct {
00133     rs_resource_descriptor_t resource_descriptor;
00134     rs_cluster_t first_cluster;
00135     rs_cluster_t current_cluster;
00136     uint8_t cluster_offset;
00137     uint16_t size;
00138     uint16_t current_position;
00139     uint8_t flags;
00140 } rs_resource_t;
00141
00142 typedef struct {
00143     uint8_t driver_mounted;
00144 } rs_global_flags_t;
00145
00146 extern rs_global_flags_t rs_global_flags;
00147
00148 // Format a device
00149 rs_op_result_t rs_format(rs_t *rs);
00150
00151 // Register a work area
00152 rs_op_result_t rs_mount(rs_driver_t driver,
00153     rs_t *rs, rs_mount_options_t options);
00154
00155 // Unregister a work area
00156 rs_op_result_t rs_umount(rs_t *rs);
00157
00158 // Open/Create a resource (you must give a empty resource)
00159 rs_op_result_t rs_open(rs_t *rs, rs_resource_code_t
00160     resource_code, rs_resource_t *resource, rs_open_resource_options_t options);
00161
00162 // Close a resource
00163 rs_op_result_t rs_close(rs_t *rs, rs_resource_t *resource);
00164
00165 // Read a byte from resource
00166 uint8_t rs_read(rs_t *rs, rs_resource_t *resource);
00167
00168 // Write a byte from resource
00169 rs_op_result_t rs_write(rs_t *rs, rs_resource_t *resource, uint8_t
00170     data_to_write);
00171
00172 // Move read/write pointer, (Expand resource size not implemented yet)
00173 rs_op_result_t rs_seek(rs_t *rs, rs_resource_t *resource,
00174     rs_seek_origin_t origin, rs_seek_int_t offset);
00175
00176 // Truncate resource size
00177 rs_op_result_t rs_truncate(rs_t *rs, rs_resource_t *resource);
00178
00179 // Flush cached data
00180 void rs_sync(rs_t *rs, rs_resource_t *resource);
00181
00182 // Get descriptor status
00183 void rs_stat(rs_t *rs, rs_resource_t *resource,
00184     rs_stat_t *stat);
00185
00186 // Rewind the position of a resource pointer
00187 rs_op_result_t rs_rewind(rs_t *rs, rs_resource_t *resource);
00188
00189 // Create/Allocate a new resource if available
00190 rs_resource_code_t rs_alloc(rs_t *rs);
00191
00192 // Make a resource free to be allocated for another one
00193 uint8_t rs_release(rs_t *rs, rs_resource_t *resource);
00194
00195 // Get size of a resource
00196 rs_resource_size_t rs_size(rs_resource_t *resource);
00197
00198 // Get the current read/write pointer
00199 rs_resource_size_t rs_tell(rs_resource_t *resource);
00200
00201 // Test for end-of-resource on a resource
00202 uint8_t rs_eor(rs_resource_t *resource);
00203
00204 // Test for an error on a resource
00205 uint8_t rs_error(rs_resource_t *resource);
00206
00207 // Return the current available space in the partition
00208 rs_resource_size_t rs_available_space(rs_t *rs);
00209
00210 // Return the total space in the partition

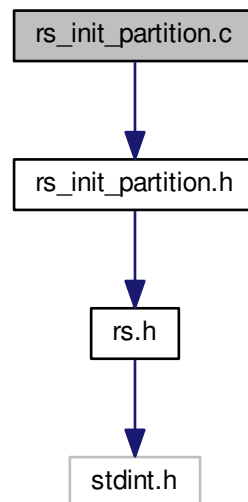
```

```
00206 rs_resource_size_t rs_total_space(rs_t *rs);  
00207  
00208 #endif // __SDCC_RS_H__
```

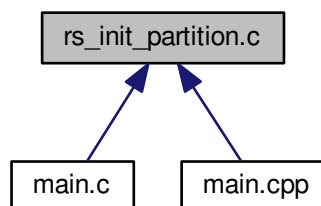
5.15 rs_init_partition.c File Reference

```
#include "rs_init_partition.h"
```

Include dependency graph for rs_init_partition.c:



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



Macros

- `#define __SDCC_RS_INIT_PARTITION_C__ 1`

Functions

- `void rs_init_partition(rs_t *rs, rs_disk_size_t size, rs_environment_t env)`

5.15.1 Macro Definition Documentation

5.15.1.1 `#define __SDCC_RS_INIT_PARTITION_C__ 1`

SDCC - PIC resource system.

[rs_init_partition.c](#)

Initializes a rs partition

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [rs_init_partition.c](#).

5.15.2 Function Documentation

5.15.2.1 `void rs_init_partition (rs_t *rs, rs_disk_size_t size, rs_environment_t env)`

Definition at line 16 of file [rs_init_partition.c](#).

5.16 [rs_init_partition.c](#)

```

00001
00011 #ifndef __SDCC_RS_INIT_PARTITION_C__
00012 #define __SDCC_RS_INIT_PARTITION_C__ 1
00013
00014 #include "rs_init_partition.h"
00015
00016 void rs_init_partition(rs_t *rs, rs_disk_size_t size,
00017                      rs_environment_t env) {
00018     switch(size) {
00019
00020         case RS_DISK_32K:
00021             if (env == RS_ENV_VIRTUAL) {
00022                 rs->driver = RS_DRIVER_VIRTUAL;
00023             } else {
00024                 rs->driver = RS_DRIVER_ARDUINO_EEPROM;
00025             }
00026             rs->memory_size = 0x7f94; //32660;
00027             rs->resource_descriptor_table_address = 0x0020; //32;
00028             rs->cluster_table_address = 0x00a0; //160;
00029             rs->sizeof_resource_descriptor_table = 0x0080; //128;
00030             rs->sizeof_cluster_table = 0x7ef4; //32500;
00031             rs->sizeof_resource_descriptor = 0x04; //4;
00032             rs->sizeof_cluster = 0x82; //130;
00033             rs->resource_descriptor_count = 0x20; //32;
00034             rs->cluster_count = 0xfa; //250;
00035             rs->sizeof_cluster_data = 0x80; //128;
00036             rs->sizeof_cluster_control = 0x02; //2;
00037             rs->free_clusters = 0xfa; //250;
00038             rs->flags = 0x00; //0;
00039             break;
00040
00041         case RS_DISK_24K:
00042             if (env == RS_ENV_VIRTUAL) {
00043                 rs->driver = RS_DRIVER_VIRTUAL;
00044             } else {
00045                 rs->driver = RS_DRIVER_ARDUINO_EEPROM;
00046             }
00047             rs->memory_size = 0x5f96; //24470;
00048             rs->resource_descriptor_table_address = 0x20; //32;
00049             rs->cluster_table_address = 0xa0; //160;
00050             rs->sizeof_resource_descriptor_table = 0x80; //128;
00051             rs->sizeof_cluster_table = 0x5ef6; //24310;
00052             rs->sizeof_resource_descriptor = 0x4; //4;
00053             rs->sizeof_cluster = 0x82; //130;
00054             rs->resource_descriptor_count = 0x20; //32;
00055             rs->cluster_count = 0xbb; //187;
00056             rs->sizeof_cluster_data = 0x80; //128;
00057             rs->sizeof_cluster_control = 0x2; //2;
00058             rs->free_clusters = 0xbb; //187;
00059             rs->flags = 0x00; //0;

```

```

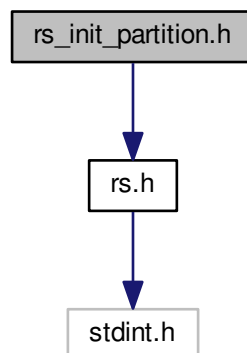
00060         break;
00061
00062     case RS_DISK_8K:
00063         if (env == RS_ENV_VIRTUAL) {
00064             rs->driver = RS_DRIVER_VIRTUAL;
00065         } else {
00066             rs->driver = RS_DRIVER_ARDUINO_EEPROM;
00067         }
00068         rs->memory_size = 0x2000; //8192;
00069         rs->resource_descriptor_table_address = 0x0020; //32;
00070         rs->cluster_table_address = 0x00a0; //160;
00071         rs->sizeof_resource_descriptor_table = 0x0080; //128;
00072         rs->sizeof_cluster_table = 0x1f60; //8032;
00073         rs->sizeof_resource_descriptor = 0x04; //4;
00074         rs->sizeof_cluster = 0x20; //32;
00075         rs->resource_descriptor_count = 0x20; //32;
00076         rs->cluster_count = 0xfb; //251;
00077         rs->sizeof_cluster_data = 0x1e; //30;
00078         rs->sizeof_cluster_control = 0x02; //2;
00079         rs->free_clusters = 0xfb; //251;
00080         rs->flags = 0x00; //0;
00081     break;
00082
00083     default:
00084
00085         if (env == RS_ENV_VIRTUAL) {
00086             rs->driver = RS_DRIVER_VIRTUAL;
00087         } else {
00088             rs->driver = RS_DRIVER_ARDUINO_EEPROM;
00089         }
00090         rs->memory_size = 0xf46; //3910;
00091         rs->resource_descriptor_table_address = 0x0020; //32;
00092         rs->cluster_table_address = 0x00a0; //160;
00093         rs->sizeof_resource_descriptor_table = 0x0080; //128;
00094         rs->sizeof_cluster_table = 0xea6; //3750;
00095         rs->sizeof_resource_descriptor = 0x04; //4;
00096         rs->sizeof_cluster = 0xf; //32;
00097         rs->resource_descriptor_count = 0x20; //32;
00098         rs->cluster_count = 0xfa; //250;
00099         rs->sizeof_cluster_data = 0xd; //13;
00100         rs->sizeof_cluster_control = 0x02; //2;
00101         rs->free_clusters = 0xfa; //250;
00102         rs->flags = 0x00; //0;
00103     break;
00104 }
00105 }
00106
00107 #endif // __SDCC_RS_INIT_PARTITION_C__

```

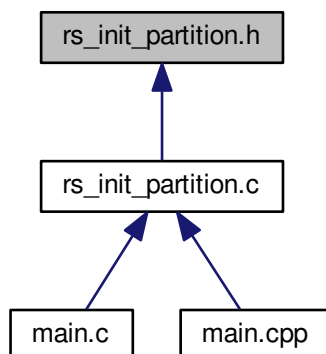
5.17 rs_init_partition.h File Reference

#include "rs.h"

Include dependency graph for rs_init_partition.h:



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



Enumerations

- enum [rs_disk_size_t](#) { [RS_DISK_4K](#), [RS_DISK_8K](#), [RS_DISK_24K](#), [RS_DISK_32K](#) }
- enum [rs_environment_t](#) { [RS_ENV_ARDUINO](#), [RS_ENV_VIRTUAL](#) }

Functions

- void [rs_init_partition](#) ([rs_t](#) *rs, [rs_disk_size_t](#) size, [rs_environment_t](#) env)

5.17.1 Enumeration Type Documentation

5.17.1.1 enum [rs_disk_size_t](#)

SDCC - PIC resource system.

[rs_init_partition.h](#)

Initializes a rs partition

Author

Dalmir da Silva dalmirdasilva@gmail.com

Enumerator

[RS_DISK_4K](#)

[RS_DISK_8K](#)

[RS_DISK_24K](#)

[RS_DISK_32K](#)

Definition at line [16](#) of file [rs_init_partition.h](#).

5.17.1.2 enum rs_environment_t

Enumerator

RS_ENV_ARDUINO***RS_ENV_VIRTUAL***Definition at line 23 of file [rs_init_partition.h](#).

5.17.2 Function Documentation

5.17.2.1 void rs_init_partition (rs_t *rs, rs_disk_size_t size, rs_environment_t env)

Definition at line 16 of file [rs_init_partition.c](#).

5.18 rs_init_partition.h

```

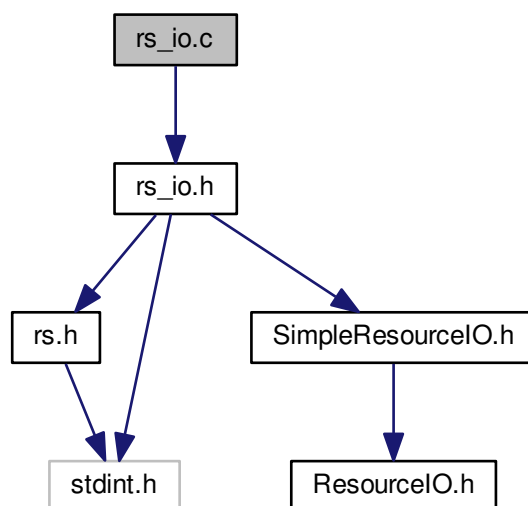
00001
00011 #ifndef __SDCC_RS_INIT_PARTITION_H__
00012 #define __SDCC_RS_INIT_PARTITION_H__ 1
00013
00014 #include "rs.h"
00015
00016 typedef enum {
00017     RS_DISK_4K,
00018     RS_DISK_8K,
00019     RS_DISK_24K,
00020     RS_DISK_32K
00021 } rs_disk_size_t;
00022
00023 typedef enum {
00024     RS_ENV_ARDUINO,
00025     RS_ENV_VIRTUAL
00026 } rs_environment_t;
00027
00028 void rs_init_partition(rs_t *rs, rs_disk_size_t size,
00029     rs_environment_t env);
00029
00030 #endif /* __SDCC_RS_INIT_PARTITION_H__ */

```

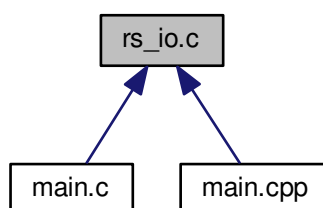
5.19 rs_io.c File Reference

```
#include "rs_io.h"
```

Include dependency graph for rs_io.c:



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



Macros

- `#define __SDCC_RS_IO_C__ 1`

Functions

- `uint8_t _rs_io_read(rs_driver_t driver, rs_memory_address_t address)`
- `void _rs_io_write(rs_driver_t driver, rs_memory_address_t address, uint8_t data)`

5.19.1 Macro Definition Documentation

5.19.1.1 #define __SDCC_RS_IO_C__ 1

SDCC - PIC resource system.

[rs_io.c](#)

IO lib for rs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [rs_io.c](#).

5.19.2 Function Documentation

5.19.2.1 uint8_t _rs_io_read (rs_driver_t driver, rs_memory_address_t address)

SDCC - PIC resource system.

[rs_io.h](#)

IO lib for rs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 16 of file [rs_io.c](#).

5.19.2.2 void _rs_io_write (rs_driver_t driver, rs_memory_address_t address, uint8_t data)

Definition at line 20 of file [rs_io.c](#).

5.20 rs_io.c

```

00001
00011 #ifndef __SDCC_RS_IO_C__
00012 #define __SDCC_RS_IO_C__ 1
00013
00014 #include "rs_io.h"
00015
00016 uint8_t _rs_io_read(rs_driver_t driver,
00017                    rs_memory_address_t address) {
00018     return SimpleResourceIO::getAssociatedIO(driver)->
00019     read(address);
00018 }
00019
00020 void _rs_io_write(rs_driver_t driver, rs_memory_address_t address
00021 , uint8_t data) {
00022     SimpleResourceIO::getAssociatedIO(driver)->
00023     write(address, data);
00022 }
00023
00024 #endif // __SDCC_RS_IO_C__

```

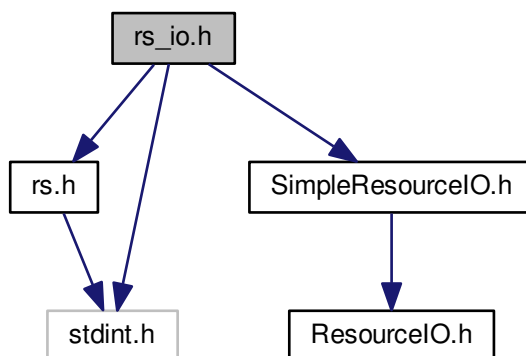
5.21 rs_io.h File Reference

```

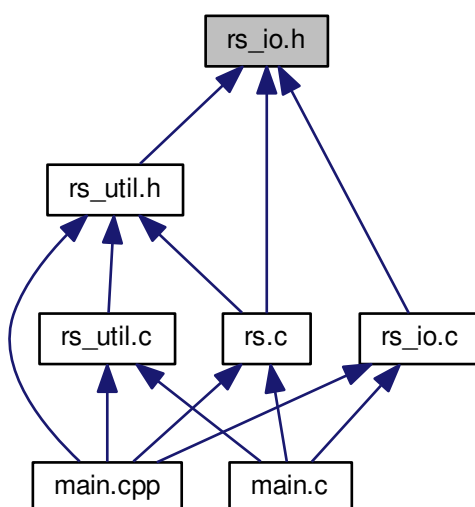
#include "rs.h"
#include <stdint.h>
#include <SimpleResourceIO.h>

```

Include dependency graph for `rs_io.h`:



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



Functions

- `uint8_t _rs_io_read(rs_driver_t driver, rs_memory_address_t address)`
- `void _rs_io_write(rs_driver_t driver, rs_memory_address_t address, uint8_t data)`

5.21.1 Function Documentation

5.21.1.1 `uint8_t rs_io_read (rs_driver_t driver, rs_memory_address_t address)`

SDCC - PIC resource system.

[rs_io.h](#)

IO lib for rs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 16 of file [rs_io.c](#).

5.21.1.2 `void rs_io_write (rs_driver_t driver, rs_memory_address_t address, uint8_t data)`

Definition at line 20 of file [rs_io.c](#).

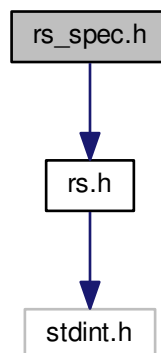
5.22 rs_io.h

```
00001
00011 #ifndef __SDCC_RS_IO_H__
00012 #define __SDCC_RS_IO_H__ 1
00013
00014 #include "rs.h"
00015 #include <stdint.h>
00016 #include <SimpleResourceIO.h>
00017
00018 uint8_t _rs_io_read(rs_driver_t driver,
00019                     rs_memory_address_t address);
00019
00020 void _rs_io_write(rs_driver_t driver, rs_memory_address_t address
00021                  , uint8_t data);
00021
00022 #endif // __SDCC_RS_IO_H__
```

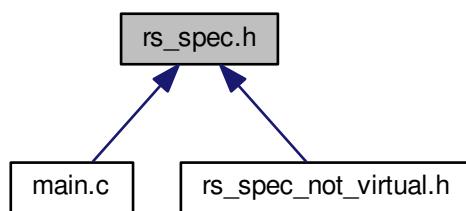
5.23 rs_spec.h File Reference

`#include "rs.h"`

Include dependency graph for rs_spec.h:



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



Macros

- `#define RS_SPEC_DRIVER RS_DRIVER_EXTERNAL_EEPROM`
- `#define rs_spec_printf printf`

Functions

- void `format_spec (rs_t *rs)`
- void `mount_spec (rs_t *rs)`
- void `umount_spec (rs_t *rs)`
- void `alloc_resource_spec (rs_t *rs)`
- void `try_to_alloc_resources_that_is_possible_spec (rs_t *rs)`
- void `open_resource_spec (rs_t *rs)`
- void `write_resource_spec (rs_t *rs)`
- void `rewind_resource_spec (rs_t *rs)`
- void `read_resource_spec (rs_t *rs)`
- void `close_resource_spec (rs_t *rs)`
- void `try_read_when_end_of_resource_is_reached_spec (rs_t *rs)`
- void `try_read_when_resource_is_closed_spec (rs_t *rs)`
- void `seek_resource_spec (rs_t *rs)`
- void `random_read_resource_spec (rs_t *rs)`
- void `random_read_with_seek_resource_spec (rs_t *rs)`
- void `random_read_with_seek_opening_resource_spec (rs_t *rs)`
- void `size_resource_spec (rs_t *rs)`
- void `tell_resource_spec (rs_t *rs)`
- void `tell_with_seek_resource_spec (rs_t *rs)`
- void `total_space_resource_spec (rs_t *rs)`
- void `allocating_multi_format_spec (rs_t *rs)`
- void `read_only_mounting_spec (rs_t *rs)`
- void `read_only_opening_spec (rs_t *rs)`

5.23.1 Macro Definition Documentation

5.23.1.1 `#define RS_SPEC_DRIVER RS_DRIVER_EXTERNAL_EEPROM`

Definition at line 6 of file [rs_spec.h](#).

5.23.1.2 #define rs_spec_printf printf

Definition at line 10 of file [rs_spec.h](#).

5.23.2 Function Documentation

5.23.2.1 void alloc_resource_spec (rs_t * rs)

Definition at line 61 of file [rs_spec.h](#).

5.23.2.2 void allocating_multi_format_spec (rs_t * rs)

Definition at line 526 of file [rs_spec.h](#).

5.23.2.3 void close_resource_spec (rs_t * rs)

Definition at line 191 of file [rs_spec.h](#).

5.23.2.4 void format_spec (rs_t * rs)

Definition at line 15 of file [rs_spec.h](#).

5.23.2.5 void mount_spec (rs_t * rs)

Definition at line 29 of file [rs_spec.h](#).

5.23.2.6 void open_resource_spec (rs_t * rs)

Definition at line 102 of file [rs_spec.h](#).

5.23.2.7 void random_read_resource_spec (rs_t * rs)

Definition at line 284 of file [rs_spec.h](#).

5.23.2.8 void random_read_with_seek_opening_resource_spec (rs_t * rs)

Definition at line 365 of file [rs_spec.h](#).

5.23.2.9 void random_read_with_seek_resource_spec (rs_t * rs)

Definition at line 323 of file [rs_spec.h](#).

5.23.2.10 void read_only_mounting_spec (rs_t * rs)

Definition at line 559 of file [rs_spec.h](#).

5.23.2.11 void read_only_opening_spec (rs_t * rs)

Definition at line 587 of file [rs_spec.h](#).

5.23.2.12 void read_resource_spec (rs_t * rs)

Definition at line 165 of file [rs_spec.h](#).

5.23.2.13 void rewind_resource_spec (rs_t * rs)

Definition at line 143 of file [rs_spec.h](#).

5.23.2.14 void seek_resource_spec (rs_t * rs)

Definition at line 259 of file [rs_spec.h](#).

5.23.2.15 void size_resource_spec (rs_t * rs)

Definition at line 411 of file [rs_spec.h](#).

5.23.2.16 void tell_resource_spec (rs_t * rs)

Definition at line 436 of file [rs_spec.h](#).

5.23.2.17 void tell_with_seek_resource_spec (rs_t * rs)

Definition at line 460 of file [rs_spec.h](#).

5.23.2.18 void total_space_resource_spec (rs_t * rs)

Definition at line 499 of file [rs_spec.h](#).

5.23.2.19 void try_read_when_end_of_resource_is_reached_spec (rs_t * rs)

Definition at line 213 of file [rs_spec.h](#).

5.23.2.20 void try_read_when_resource_is_closed_spec (rs_t * rs)

Definition at line 237 of file [rs_spec.h](#).

5.23.2.21 void try_to_alloc_resources_that_is_possible_spec (rs_t * rs)

Definition at line 79 of file [rs_spec.h](#).

5.23.2.22 void umount_spec (rs_t * rs)

Definition at line 45 of file [rs_spec.h](#).

5.23.2.23 void write_resource_spec (rs_t * rs)

Definition at line 122 of file [rs_spec.h](#).

5.24 rs_spec.h

```
00001 #include "rs.h"
00002
00003 #if VIRTUAL_ENVIROMENT == 1
00004 #define RS_SPEC_DRIVER RS_DRIVER_VIRTUAL
00005 #else
00006 #define RS_SPEC_DRIVER RS_DRIVER_EXTERNAL_EEPROM
00007 #endif
00008
00009 #ifndef rs_spec_printf
00010 #define rs_spec_printf printf
00011 #endif
00012
00013 #ifndef RS_SPEC_IGNORE_0
00014
00015 void format_spec(rs_t *rs) {
00016     rs_op_result_t op_r;
00017     rs_init_partition(rs, RS_DISK_32K,
00018         RS_ENV_VIRTUAL);
00019     op_r = rs_format(rs);
00020     if (op_r != RS_OP_RESULT_SUCCESS) {
00021         rs_spec_printf("(F) fomat spec failed. error: %d\n", op_r);
00022     } else {
00023         rs_spec_printf("(*) fomat spec passed.\n",
00024             RS_OP_RESULT_SUCCESS);
00025     }
00026 }
00027 #endif
00028
00029 #ifndef RS_SPEC_IGNORE_1
00030 void mount_spec(rs_t *rs) {
00031     rs_op_result_t op_r;
00032     rs_init_partition(rs, RS_DISK_32K,
```

```

    RS_ENV_VIRTUAL);
00032     op_r = rs_format(rs);
00033     op_r = rs_mount(RS_SPEC_DRIVER, rs,
    RS_MOUNT_OPTION_NORMAL);
00034     if (op_r != RS_OP_RESULT_SUCCESS) {
00035         rs_spec_printf("(F) mount spec failed. error: %d\n", op_r);
00036     } else {
00037         rs_spec_printf("(*) mount spec passed.\n",
    RS_OP_RESULT_SUCCESS);
00038     }
00039     rs_umount(rs);
00040 }
00041 #endif
00042
00043 #ifndef RS_SPEC_IGNORE_2
00044
00045 void umount_spec(rs_t *rs) {
00046     rs_op_result_t op_r;
00047     rs_init_partition(rs, RS_DISK_32K,
    RS_ENV_VIRTUAL);
00048     op_r = rs_format(rs);
00049     op_r = rs_mount(RS_SPEC_DRIVER, rs,
    RS_MOUNT_OPTION_NORMAL);
00050     op_r = rs_umount(rs);
00051     if (op_r != RS_OP_RESULT_SUCCESS) {
00052         rs_spec_printf("(F) umount spec failed. error: %d\n", op_r);
00053     } else {
00054         rs_spec_printf("(*) umount spec passed.\n",
    RS_OP_RESULT_SUCCESS);
00055     }
00056 }
00057 #endif
00058
00059 #ifndef RS_SPEC_IGNORE_3
00060
00061 void alloc_resource_spec(rs_t *rs) {
00062     rs_op_result_t op_r;
00063     rs_resource_code_t rs_resource_code;
00064     rs_init_partition(rs, RS_DISK_32K,
    RS_ENV_VIRTUAL);
00065     op_r = rs_format(rs);
00066     op_r = rs_mount(RS_SPEC_DRIVER, rs,
    RS_MOUNT_OPTION_NORMAL);
00067     rs_resource_code = rs_alloc(rs);
00068     if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
00069         rs_spec_printf("(F) alloc_resource spec failed. error: %d\n", op_r);
00070     } else {
00071         rs_spec_printf("(*) alloc_resource spec passed.\n",
    RS_OP_RESULT_SUCCESS);
00072     }
00073     rs_umount(rs);
00074 }
00075 #endif
00076
00077 #ifndef RS_SPEC_IGNORE_4
00078
00079 void try_to_alloc_resources_that_is_possible_spec(
    rs_t *rs) {
00080     rs_op_result_t op_r;
00081     rs_resource_code_t rs_resource_code[2];
00082     uint8_t i = 0;
00083     rs_init_partition(rs, RS_DISK_32K,
    RS_ENV_VIRTUAL);
00084     op_r = rs_format(rs);
00085     op_r = rs_mount(RS_SPEC_DRIVER, rs,
    RS_MOUNT_OPTION_NORMAL);
00086     for (; i < rs->resource_descriptor_count; i++) {
00087         rs_resource_code[0] = rs_alloc(rs);
00088     }
00089     rs_resource_code[1] = rs_alloc(rs);
00090     if (rs_resource_code[0] == (rs->resource_descriptor_count - 1) &&
    rs_resource_code[1] == RS_NULL_RESOURCE_CODE) {
00091         rs_spec_printf("(*) try_to_alloc_resources_that_is_possible spec passed.\n",
    RS_OP_RESULT_SUCCESS);
00092     } else {
00093         rs_spec_printf("(F) try_to_alloc_resources_that_is_possible spec failed. error: %x\n",
    rs_resource_code[0]);
00094         while (1);
00095     }
00096     rs_umount(rs);
00097 }
00098 #endif
00099
00100 #ifndef RS_SPEC_IGNORE_5
00101
00102 void open_resource_spec(rs_t *rs) {
00103     rs_op_result_t op_r;

```

```

00104     rs_resource_code_t rs_resource_code;
00105     rs_resource_t resource;
00106     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00107     op_r = rs_format(rs);
00108     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00109     rs_resource_code = rs_alloc(rs);
00110     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00111     if (op_r != RS_OP_RESULT_SUCCESS) {
00112         rs_spec_printf("(F) open_resource spec failed. error: %d\n", op_r);
00113     } else {
00114         rs_spec_printf("(*) open_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00115     }
00116     rs_umount(rs);
00117 }
00118 #endif
00119
00120 #ifndef RS_SPEC_IGNORE_6
00121
00122 void write_resource_spec(rs_t *rs) {
00123     rs_op_result_t op_r;
00124     rs_resource_code_t rs_resource_code;
00125     rs_resource_t resource;
00126     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00127     op_r = rs_format(rs);
00128     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00129     rs_resource_code = rs_alloc(rs);
00130     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00131     op_r = rs_write(rs, &resource, 0xaa);
00132     if (op_r != RS_OP_RESULT_SUCCESS) {
00133         rs_spec_printf("(F) write_resource spec failed. error: %d\n", op_r);
00134     } else {
00135         rs_spec_printf("(*) write_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00136     }
00137     rs_umount(rs);
00138 }
00139 #endif
00140
00141 #ifndef RS_SPEC_IGNORE_7
00142
00143 void rewind_resource_spec(rs_t *rs) {
00144     rs_op_result_t op_r;
00145     rs_resource_code_t rs_resource_code;
00146     rs_resource_t resource;
00147     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00148     op_r = rs_format(rs);
00149     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00150     rs_resource_code = rs_alloc(rs);
00151     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00152     op_r = rs_write(rs, &resource, 0xAA);
00153     op_r = rs_rewind(rs, &resource);
00154     if (op_r != RS_OP_RESULT_SUCCESS) {
00155         rs_spec_printf("(F) rewind_resource spec failed. error: %d\n", op_r);
00156     } else {
00157         rs_spec_printf("(*) rewind_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00158     }
00159     rs_umount(rs);
00160 }
00161 #endif
00162
00163 #ifndef RS_SPEC_IGNORE_8
00164
00165 void read_resource_spec(rs_t *rs) {
00166     rs_op_result_t op_r;
00167     rs_resource_code_t rs_resource_code;
00168     rs_resource_t resource;
00169     unsigned char c[2];
00170     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00171     op_r = rs_format(rs);
00172     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00173     rs_resource_code = rs_alloc(rs);
00174     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00175     op_r = rs_write(rs, &resource, 0x41);

```



```

00176     op_r = rs_write(rs, &resource, 0xA1);
00177     op_r = rs_rewind(rs, &resource);
00178     c[0] = rs_read(rs, &resource);
00179     c[1] = rs_read(rs, &resource);
00180     if (c[0] != 0xA1 || c[1] != 0xA1) {
00181         rs_spec_printf("(F) read_resource spec failed. error: %x\n", c[0]);
00182     } else {
00183         rs_spec_printf("(*) read_resource spec passed.\n",
00184             RS_OP_RESULT_SUCCESS);
00185     }
00186     rs_umount(rs);
00187 }
00188 #endif
00189 #ifndef RS_SPEC_IGNORE_9
00190
00191 void close_resource_spec(rs_t *rs) {
00192     rs_op_result_t op_r;
00193     rs_resource_code_t rs_resource_code;
00194     rs_resource_t resource;
00195     unsigned char c[2];
00196     rs_init_partition(rs, RS_DISK_32K,
00197         RS_ENV_VIRTUAL);
00198     op_r = rs_format(rs);
00199     op_r = rs_mount(RS_SPEC_DRIVER, rs,
00200         RS_MOUNT_OPTION_NORMAL);
00201     rs_resource_code = rs_alloc(rs);
00202     op_r = rs_open(rs, rs_resource_code, &resource,
00203         RS_OPEN_RESOURCE_OPTION_NORMAL);
00204     op_r = rs_close(rs, &resource);
00205     if (op_r != RS_OP_RESULT_SUCCESS) {
00206         rs_spec_printf("(F) close_resource spec failed. error: %x\n", op_r);
00207     } else {
00208         rs_spec_printf("(*) close_resource spec passed.\n",
00209             RS_OP_RESULT_SUCCESS);
00210     }
00211     rs_umount(rs);
00212 }
00213 #endif
00214 #ifndef RS_SPEC_IGNORE_10
00215
00216 void try_read_when_end_of_resource_is_reached_spec(
00217     rs_t *rs) {
00218     rs_op_result_t op_r;
00219     rs_resource_code_t rs_resource_code;
00220     rs_resource_t resource;
00221     rs_init_partition(rs, RS_DISK_32K,
00222         RS_ENV_VIRTUAL);
00223     op_r = rs_format(rs);
00224     op_r = rs_mount(RS_SPEC_DRIVER, rs,
00225         RS_MOUNT_OPTION_NORMAL);
00226     rs_resource_code = rs_alloc(rs);
00227     op_r = rs_open(rs, rs_resource_code, &resource,
00228         RS_OPEN_RESOURCE_OPTION_NORMAL);
00229     op_r = rs_write(rs, &resource, 0xA1);
00230     op_r = rs_rewind(rs, &resource);
00231     rs_read(rs, &resource);
00232     rs_read(rs, &resource);
00233     if (op_r == 0 && (rs_eor(&resource))) {
00234         rs_spec_printf("(*) try_read_when_end_of_resource_is_reached spec passed.\n",
00235             RS_OP_RESULT_SUCCESS);
00236     } else {
00237         rs_spec_printf("(F) try_read_when_end_of_resource_is_reached spec failed. error: %x\n",
00238             op_r);
00239     }
00240     rs_umount(rs);
00241 }
00242 #endif
00243 #ifndef RS_SPEC_IGNORE_11
00244
00245 void try_read_when_resource_is_closed_spec(
00246     rs_t *rs) {
00247     rs_op_result_t op_r;
00248     rs_resource_code_t rs_resource_code;
00249     rs_resource_t resource;
00250     rs_init_partition(rs, RS_DISK_32K,
00251         RS_ENV_VIRTUAL);
00252     op_r = rs_format(rs);
00253     op_r = rs_mount(RS_SPEC_DRIVER, rs,
00254         RS_MOUNT_OPTION_NORMAL);
00255     rs_resource_code = rs_alloc(rs);
00256     op_r = rs_open(rs, rs_resource_code, &resource,
00257         RS_OPEN_RESOURCE_OPTION_NORMAL);
00258     op_r = rs_close(rs, &resource);
00259     rs_read(rs, &resource);

```

```

00248     if (op_r == 0 && (resource.flags |
RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ)) {
00249         rs_spec_printf("(*) try_read_when_resource_is_closed spec passed.\n",
RS_OP_RESULT_SUCCESS);
00250     } else {
00251         rs_spec_printf("(F) try_read_when_resource_is_closed spec failed. error: %x\n", op_r)
;
00252     }
00253     rs_umount(rs);
00254 }
00255 #endif
00256
00257 #ifndef RS_SPEC_IGNORE_12
00258
00259 void seek_resource_spec(rs_t *rs) {
00260     rs_op_result_t op_r;
00261     rs_resource_code_t rs_resource_code;
00262     rs_resource_t resource;
00263     uint8_t i = 0;
00264     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00265     op_r = rs_format(rs);
00266     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00267     rs_resource_code = rs_alloc(rs);
00268     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00269     for (; i < 50; i++) {
00270         op_r = rs_write(rs, &resource, (i + 0x65));
00271     }
00272     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
00273     if (op_r == RS_OP_RESULT_SUCCESS) {
00274         rs_spec_printf("(*) seek_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00275     } else {
00276         rs_spec_printf("(F) seek_resource spec failed. error: %x\n", op_r);
00277     }
00278     rs_umount(rs);
00279 }
00280 #endif
00281
00282 #ifndef RS_SPEC_IGNORE_13
00283
00284 void random_read_resource_spec(rs_t *rs) {
00285     rs_op_result_t op_r;
00286     rs_resource_code_t rs_resource_code;
00287     rs_resource_t resource;
00288     uint8_t i = 0;
00289     unsigned char c[5], first_write_char = 0x65;
00290     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00291     op_r = rs_format(rs);
00292     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00293     rs_resource_code = rs_alloc(rs);
00294     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00295     for (; i < 255; i++) {
00296         op_r = rs_write(rs, &resource, (i + first_write_char));
00297     }
00298     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
00299     c[0] = rs_read(rs, &resource);
00300     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 10);
00301     c[1] = rs_read(rs, &resource);
00302     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 48);
00303     c[2] = rs_read(rs, &resource);
00304     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 20);
00305     c[3] = rs_read(rs, &resource);
00306     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 0);
00307     c[4] = rs_read(rs, &resource);
00308     if (c[0] == first_write_char + 20 &&
00309         c[1] == first_write_char + 31 &&
00310         c[2] == first_write_char + 48 &&
00311         c[3] == first_write_char + 69 &&
00312         c[4] == first_write_char + 0) {
00313         rs_spec_printf("(*) random_read_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00314     } else {
00315         rs_spec_printf("(F) random_read_resource spec failed. error: %x\n", op_r);
00316     }
00317     rs_umount(rs);
00318 }
00319 #endif
00320
00321 #ifndef RS_SPEC_IGNORE_14
00322
00323 void random_read_with_seek_resource_spec(rs_t *rs) {

```

```

00324     rs_op_result_t op_r;
00325     rs_resource_code_t rs_resource_code;
00326     rs_resource_t resource;
00327     uint8_t i = 0;
00328     unsigned char c[255];
00329     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00330     op_r = rs_format(rs);
00331     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00332     rs_resource_code = rs_alloc(rs);
00333     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00334     for (i = 0; i < 255; i++) {
00335         op_r = rs_write(rs, &resource, i);
00336     }
00337     rs_rewind(rs, &resource);
00338
00339     for (i = 0; i < 255; i++) {
00340         rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, i);
00341         c[i] = rs_read(rs, &resource);
00342         rs_read(rs, &resource);
00343         rs_read(rs, &resource);
00344         rs_read(rs, &resource);
00345     }
00346
00347     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 199);
00348     if ((i = rs_read(rs, &resource)) != 199) {
00349         rs_spec_printf("(F) random_read_with_seek_resource_spec spec failed. != 199\n", 0);
00350     }
00351
00352     for (i = 0; i < 255; i++) {
00353         if (i != c[i]) {
00354             rs_spec_printf("(F) random_read_with_seek_resource_spec spec failed. error: %x\n"
, i);
00355         }
00356     }
00357
00358     rs_spec_printf("(*) random_read_with_seek_resource_spec spec passed.\n",
RS_OP_RESULT_SUCCESS);
00359     rs_umount(rs);
00360 }
00361 #endif
00362
00363 #ifndef RS_SPEC_IGNORE_15
00364
00365 void random_read_with_seek_opening_resource_spec(
rs_t *rs) {
00366     rs_op_result_t op_r;
00367     rs_resource_code_t rs_resource_code;
00368     rs_resource_t resource;
00369     uint8_t i = 0;
00370     unsigned char c[255];
00371     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00372     op_r = rs_format(rs);
00373     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00374     rs_resource_code = rs_alloc(rs);
00375     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00376     for (i = 0; i < 255; i++) {
00377         op_r = rs_write(rs, &resource, i);
00378     }
00379     rs_close(rs, &resource);
00380
00381     for (i = 0; i < 255; i++) {
00382         op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00383         rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, i);
00384         c[i] = rs_read(rs, &resource);
00385         rs_read(rs, &resource);
00386         rs_read(rs, &resource);
00387         rs_read(rs, &resource);
00388         rs_close(rs, &resource);
00389     }
00390
00391     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00392     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 199);
00393     if ((i = rs_read(rs, &resource)) != 199) {
00394         rs_spec_printf("(F) random_read_with_seek_opening_resource_spec spec failed. != 199\n"
, op_r);
00395     }
00396     rs_close(rs, &resource);
00397
00398     for (i = 0; i < 255; i++) {

```

```

00399         if (i != c[i]) {
00400             rs_spec_printf("(F) random_read_with_seek_opening_resource_spec spec failed.
error: %x\n", c[i]);
00401         }
00402     }
00403
00404     rs_spec_printf("(*) random_read_with_seek_opening_resource_spec spec passed.\n",
RS_OP_RESULT_SUCCESS);
00405     rs_umount(rs);
00406 }
00407 #endif
00408
00409 #ifndef RS_SPEC_IGNORE_16
00410
00411 void size_resource_spec(rs_t *rs) {
00412     rs_op_result_t op_r;
00413     rs_resource_code_t rs_resource_code;
00414     rs_resource_t resource;
00415     uint16_t i = 0;
00416     uint16_t size = 0xf40;
00417     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00418     op_r = rs_format(rs);
00419     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00420     rs_resource_code = rs_alloc(rs);
00421     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00422     for (; i < size; i++) {
00423         op_r = rs_write(rs, &resource, 0x65);
00424     }
00425     if (rs_size(&resource) == 0xf40) {
00426         rs_spec_printf("(*) size_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00427     } else {
00428         rs_spec_printf("(F) size_resource spec failed. error: %x\n", size);
00429     }
00430     rs_umount(rs);
00431 }
00432 #endif
00433
00434 #ifndef RS_SPEC_IGNORE_17
00435
00436 void tell_resource_spec(rs_t *rs) {
00437     rs_op_result_t op_r;
00438     rs_resource_code_t rs_resource_code;
00439     rs_resource_t resource;
00440     uint8_t i = 0;
00441     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00442     op_r = rs_format(rs);
00443     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00444     rs_resource_code = rs_alloc(rs);
00445     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00446     for (; i < 50; i++) {
00447         op_r = rs_write(rs, &resource, 0x65);
00448     }
00449     if (rs_tell(&resource) == 50) {
00450         rs_spec_printf("(*) tell_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
00451     } else {
00452         rs_spec_printf("(F) tell_resource spec failed. error: %x\n", op_r);
00453     }
00454     rs_umount(rs);
00455 }
00456 #endif
00457
00458 #ifndef RS_SPEC_IGNORE_18
00459
00460 void tell_with_seek_resource_spec(rs_t *rs) {
00461     rs_op_result_t op_r;
00462     rs_resource_code_t rs_resource_code;
00463     rs_resource_t resource;
00464     uint8_t i = 0;
00465     rs_resource_size_t s[5];
00466     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00467     op_r = rs_format(rs);
00468     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00469     rs_resource_code = rs_alloc(rs);
00470     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00471     for (; i < 50; i++) {
00472         op_r = rs_write(rs, &resource, 0x65);

```

```

00473     }
00474     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
00475     s[0] = rs_tell(&resource);
00476     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 10);
00477     s[1] = rs_tell(&resource);
00478     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 48);
00479     s[2] = rs_tell(&resource);
00480     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 20);
00481     s[3] = rs_tell(&resource);
00482     op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 0);
00483     s[4] = rs_tell(&resource);
00484     if (s[0] == 20 &&
00485         s[1] == 30 &&
00486         s[2] == 48 &&
00487         s[3] == 17 &&
00488         s[4] == 0) {
00489         rs_spec_printf("(*) tell_with_seek_resource spec passed.\n",
00490             RS_OP_RESULT_SUCCESS);
00491     } else {
00492         rs_spec_printf("(F) tell_with_seek_resource spec failed. error: %d\n", s[3]);
00493     }
00494     rs_umount(rs);
00495 #endif
00496
00497 #ifndef RS_SPEC_IGNORE_19
00498
00499 void total_space_resource_spec(rs_t *rs) {
00500     rs_op_result_t op_r;
00501     rs_resource_code_t rs_resource_code;
00502     rs_resource_t resource;
00503     rs_resource_size_t total_space[2];
00504     uint16_t i = 0;
00505     rs_init_partition(rs, RS_DISK_32K,
00506         RS_ENV_VIRTUAL);
00507     op_r = rs_format(rs);
00508     op_r = rs_mount(RS_SPEC_DRIVER, rs,
00509         RS_MOUNT_OPTION_NORMAL);
00510     total_space[0] = rs_available_space(rs);
00511     rs_resource_code = rs_alloc(rs);
00512     op_r = rs_open(rs, rs_resource_code, &resource,
00513         RS_OPEN_RESOURCE_OPTION_NORMAL);
00514     for (; i < rs->sizeof_cluster_data + 1; i++) {
00515         op_r = rs_write(rs, &resource, 0x65);
00516     }
00517     total_space[1] = rs_available_space(rs);
00518     if (total_space[0] - total_space[1] == (rs->sizeof_cluster_data * 2)) {
00519         rs_spec_printf("(*) total_space_resource spec passed.\n",
00520             RS_OP_RESULT_SUCCESS);
00521     } else {
00522         rs_spec_printf("(F) total_space_resource spec failed. error: %d != 50\n", total_space
00523             [0] - total_space[1]);
00524     }
00525     rs_umount(rs);
00526 }
00527 #endif
00528
00529 #ifndef RS_SPEC_IGNORE_20
00530
00531 void allocating_multi_format_spec(rs_t *rs) {
00532     rs_op_result_t op_r;
00533     rs_resource_t resource;
00534     uint8_t count = 3;
00535     uint8_t j, i;
00536     rs_resource_code_t rs_resource_code[3];
00537     uint8_t passed = 1;
00538     for (j = 0; j < count; j++) {
00539         rs_init_partition(rs, RS_DISK_32K,
00540             RS_ENV_VIRTUAL);
00541         op_r = rs_format(rs);
00542         op_r = rs_mount(RS_SPEC_DRIVER, rs,
00543             RS_MOUNT_OPTION_NORMAL);
00544         rs_resource_code[j] = rs_alloc(rs);
00545         op_r = rs_open(rs, rs_resource_code[j], &resource,
00546             RS_OPEN_RESOURCE_OPTION_NORMAL);
00547         for (i = 0; i < 50; i++) {
00548             op_r = rs_write(rs, &resource, 0x65);
00549         }
00550         rs_close(rs, &resource);
00551     }
00552     for (j = 0; j < count; j++) {
00553         if (rs_resource_code[j] != 0) {
00554             rs_spec_printf("(F) allocating_multi_format spec failed %x\n", rs_resource_code[j
00555                 ]);
00556             passed = 0;
00557         }
00558     }
00559 }

```

```

00550     if (passed) {
00551         rs_spec_printf("(*) allocating_multi_format spec passed\n",
RS_OP_RESULT_SUCCESS);
00552     }
00553     rs_umount(rs);
00554 }
00555 #endif
00556
00557 #ifndef RS_SPEC_IGNORE_21
00558
00559 void read_only_mounting_spec(rs_t *rs) {
00560     rs_op_result_t op_r;
00561     rs_resource_t resource;
00562     rs_resource_code_t rs_resource_code;
00563     char error_msg[] = "(F) read_only_mounting spec failed. %d\n";
00564     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00565     op_r = rs_format(rs);
00566     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_READ_ONLY);
00567     rs_resource_code = rs_alloc(rs);
00568     if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
00569         rs_spec_printf(error_msg, op_r);
00570     }
00571     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
00572     if (op_r != RS_OP_RESULT_SUCCESS) {
00573         rs_spec_printf(error_msg, op_r);
00574     }
00575     op_r = rs_write(rs, &resource, 0xaa);
00576     if (op_r == RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
00577         rs_spec_printf("(*) read_only_mounting spec passed.\n",
RS_OP_RESULT_SUCCESS);
00578     } else {
00579         rs_spec_printf(error_msg, op_r);
00580     }
00581     rs_umount(rs);
00582 }
00583 #endif
00584
00585 #ifndef RS_SPEC_IGNORE_22
00586
00587 void read_only_opening_spec(rs_t *rs) {
00588     rs_op_result_t op_r;
00589     rs_resource_t resource;
00590     rs_resource_code_t rs_resource_code;
00591     char error_msg[] = "(F) read_only_opening spec failed. %d\n";
00592     rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
00593     op_r = rs_format(rs);
00594     op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
00595     rs_resource_code = rs_alloc(rs);
00596     if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
00597         rs_spec_printf(error_msg, op_r);
00598     }
00599     op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_READ_ONLY);
00600     if (op_r != RS_OP_RESULT_SUCCESS) {
00601         rs_spec_printf(error_msg, op_r);
00602     }
00603     op_r = rs_write(rs, &resource, 0xaa);
00604     if (op_r == RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
00605         rs_spec_printf("(*) read_only_opening spec passed.\n",
RS_OP_RESULT_SUCCESS);
00606     } else {
00607         rs_spec_printf(error_msg, op_r);
00608     }
00609     rs_umount(rs);
00610 }
00611 #endif

```

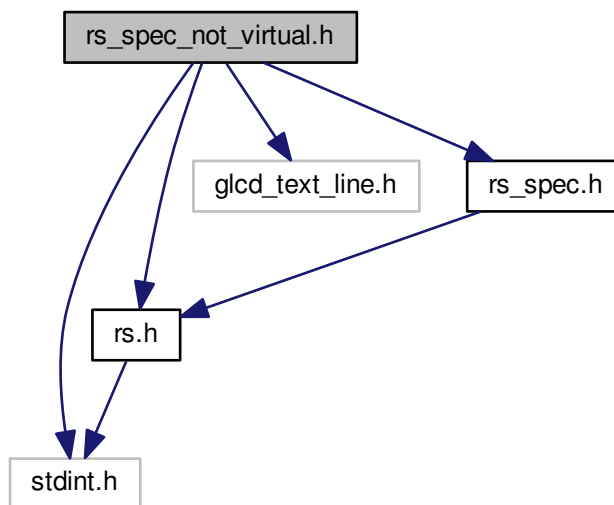
5.25 rs_spec_not_virtual.h File Reference

```

#include <stdint.h>
#include "rs.h"
#include <glcd_text_line.h>
#include "rs_spec.h"

```

Include dependency graph for rs_spec_not_virtual.h:



Macros

- `#define` [rs_spec_printf](#) `print_ln`
- `#define` [RS_SPEC_IGNORE_0](#)
- `#define` [RS_SPEC_IGNORE_1](#)
- `#define` [RS_SPEC_IGNORE_2](#)
- `#define` [RS_SPEC_IGNORE_3](#)
- `#define` [RS_SPEC_IGNORE_4](#)
- `#define` [RS_SPEC_IGNORE_5](#)
- `#define` [RS_SPEC_IGNORE_6](#)
- `#define` [RS_SPEC_IGNORE_7](#)
- `#define` [RS_SPEC_IGNORE_8](#)
- `#define` [RS_SPEC_IGNORE_9](#)
- `#define` [RS_SPEC_IGNORE_10](#)
- `#define` [RS_SPEC_IGNORE_11](#)
- `#define` [RS_SPEC_IGNORE_12](#)
- `#define` [RS_SPEC_IGNORE_13](#)
- `#define` [RS_SPEC_IGNORE_14](#)
- `#define` [RS_SPEC_IGNORE_15](#)
- `#define` [RS_SPEC_IGNORE_16](#)
- `#define` [RS_SPEC_IGNORE_17](#)
- `#define` [RS_SPEC_IGNORE_18](#)
- `#define` [RS_SPEC_IGNORE_19](#)
- `#define` [RS_SPEC_IGNORE_20](#)
- `#define` [RS_SPEC_IGNORE_21](#)
- `#define` [RS_SPEC_IGNORE_22](#)

Functions

- `void` [print_ln](#) (`char *s`, `uint8_t r`)

5.25.1 Macro Definition Documentation

5.25.1.1 `#define RS_SPEC_IGNORE_0`

Definition at line 15 of file [rs_spec_not_virtual.h](#).

5.25.1.2 `#define RS_SPEC_IGNORE_1`

Definition at line 16 of file [rs_spec_not_virtual.h](#).

5.25.1.3 `#define RS_SPEC_IGNORE_10`

Definition at line 25 of file [rs_spec_not_virtual.h](#).

5.25.1.4 `#define RS_SPEC_IGNORE_11`

Definition at line 26 of file [rs_spec_not_virtual.h](#).

5.25.1.5 `#define RS_SPEC_IGNORE_12`

Definition at line 27 of file [rs_spec_not_virtual.h](#).

5.25.1.6 `#define RS_SPEC_IGNORE_13`

Definition at line 28 of file [rs_spec_not_virtual.h](#).

5.25.1.7 `#define RS_SPEC_IGNORE_14`

Definition at line 29 of file [rs_spec_not_virtual.h](#).

5.25.1.8 `#define RS_SPEC_IGNORE_15`

Definition at line 30 of file [rs_spec_not_virtual.h](#).

5.25.1.9 `#define RS_SPEC_IGNORE_16`

Definition at line 31 of file [rs_spec_not_virtual.h](#).

5.25.1.10 `#define RS_SPEC_IGNORE_17`

Definition at line 32 of file [rs_spec_not_virtual.h](#).

5.25.1.11 `#define RS_SPEC_IGNORE_18`

Definition at line 33 of file [rs_spec_not_virtual.h](#).

5.25.1.12 `#define RS_SPEC_IGNORE_19`

Definition at line 34 of file [rs_spec_not_virtual.h](#).

5.25.1.13 `#define RS_SPEC_IGNORE_2`

Definition at line 17 of file [rs_spec_not_virtual.h](#).

5.25.1.14 `#define RS_SPEC_IGNORE_20`

Definition at line 35 of file [rs_spec_not_virtual.h](#).

5.25.1.15 `#define RS_SPEC_IGNORE_21`

Definition at line 36 of file [rs_spec_not_virtual.h](#).

5.25.1.16 #define RS_SPEC_IGNORE_22

Definition at line 37 of file [rs_spec_not_virtual.h](#).

5.25.1.17 #define RS_SPEC_IGNORE_3

Definition at line 18 of file [rs_spec_not_virtual.h](#).

5.25.1.18 #define RS_SPEC_IGNORE_4

Definition at line 19 of file [rs_spec_not_virtual.h](#).

5.25.1.19 #define RS_SPEC_IGNORE_5

Definition at line 20 of file [rs_spec_not_virtual.h](#).

5.25.1.20 #define RS_SPEC_IGNORE_6

Definition at line 21 of file [rs_spec_not_virtual.h](#).

5.25.1.21 #define RS_SPEC_IGNORE_7

Definition at line 22 of file [rs_spec_not_virtual.h](#).

5.25.1.22 #define RS_SPEC_IGNORE_8

Definition at line 23 of file [rs_spec_not_virtual.h](#).

5.25.1.23 #define RS_SPEC_IGNORE_9

Definition at line 24 of file [rs_spec_not_virtual.h](#).

5.25.1.24 #define rs_spec_printf print_ln

Definition at line 13 of file [rs_spec_not_virtual.h](#).

5.25.2 Function Documentation

5.25.2.1 void print_ln (char * s, uint8_t r)

Definition at line 6 of file [rs_spec_not_virtual.h](#).

5.26 rs_spec_not_virtual.h

```

00001 #include <stdint.h>
00002 #include "rs.h"
00003
00004 #include <glcd_text_line.h>
00005
00006 void print_ln(char *s, uint8_t r) {
00007     char b[3];
00008     i8toh(r, b);
00009     glcd_text_line_print_ln(s);
00010     glcd_text_line_print_ln(b);
00011 }
00012
00013 #define rs_spec_printf print_ln
00014
00015 #define RS_SPEC_IGNORE_0
00016 #define RS_SPEC_IGNORE_1
00017 #define RS_SPEC_IGNORE_2
00018 #define RS_SPEC_IGNORE_3
00019 #define RS_SPEC_IGNORE_4
00020 #define RS_SPEC_IGNORE_5
00021 #define RS_SPEC_IGNORE_6
00022 #define RS_SPEC_IGNORE_7
00023 #define RS_SPEC_IGNORE_8

```

```

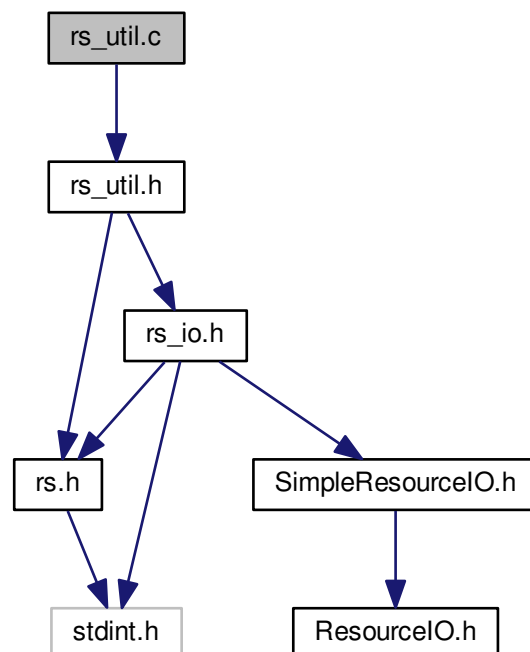
00024 #define RS_SPEC_IGNORE_9
00025 #define RS_SPEC_IGNORE_10
00026 #define RS_SPEC_IGNORE_11
00027 #define RS_SPEC_IGNORE_12
00028 #define RS_SPEC_IGNORE_13
00029 #define RS_SPEC_IGNORE_14
00030 #define RS_SPEC_IGNORE_15
00031 #define RS_SPEC_IGNORE_16
00032 #define RS_SPEC_IGNORE_17
00033 #define RS_SPEC_IGNORE_18
00034 #define RS_SPEC_IGNORE_19
00035 #define RS_SPEC_IGNORE_20
00036 #define RS_SPEC_IGNORE_21
00037 #define RS_SPEC_IGNORE_22
00038
00039 #include "rs_spec.h"

```

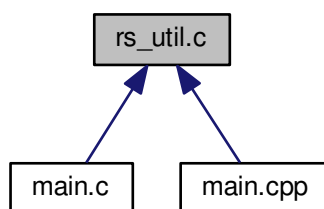
5.27 rs_util.c File Reference

```
#include "rs_util.h"
```

Include dependency graph for rs_util.c:



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



Macros

- `#define __SDCC_RS_UTIL_C__ 1`

Functions

- `void _rs_write_rs_to_disc (rs_driver_t driver, rs_t *rs)`
- `void _rs_read_rs_from_disc (rs_driver_t driver, rs_t *rs)`
- `rs_memory_address_t _rs_alloc_cluster (rs_t *rs)`
- `uint8_t _rs_is_free_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_format_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_free_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_create_cluster_chain (rs_t *rs, rs_cluster_t prev_cluster, rs_cluster_t next_cluster)`
- `void _rs_check_for_eor_reached (rs_resource_t *resource)`
- `uint8_t _rs_is_eor_reached (rs_resource_t *resource)`
- `uint8_t _rs_check_for_availability (rs_t *rs, rs_resource_t *resource)`
- `uint8_t _rs_move_current_position_ahead (rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset)`
- `uint8_t _rs_move_current_position_back (rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset)`
- `void _rs_format_resorce_descriptor (rs_t *rs, rs_resource_descriptor_t resource_descriptor)`
- `uint8_t _rs_is_driver_monted (rs_driver_t driver)`
- `void _rs_set_driver_monted (rs_driver_t driver, uint8_t is)`
- `void _rs_free_resource_descriptors (rs_t *rs)`
- `void _rs_free_resource_descriptor (rs_t *rs, rs_resource_descriptor_t resource_descriptor)`
- `void _rs_format_resource_clusters (rs_t *rs, rs_resource_t *resource)`
- `uint8_t _rs_format_clusters_chain (rs_t *rs, rs_cluster_t cluster)`
- `uint8_t _rs_has_invalid_attributes (rs_t *rs)`

5.27.1 Macro Definition Documentation

5.27.1.1 `#define __SDCC_RS_UTIL_C__ 1`

SDCC - PIC resource system.

[rs_util.c](#)

Util lib for rs

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [rs_util.c](#).

5.27.2 Function Documentation

5.27.2.1 `rs_memory_address_t rs_alloc_cluster (rs_t * rs)`

Allocate a free cluster from disc if any.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 36 of file [rs_util.c](#).

5.27.2.2 `uint8_t rs_check_for_availability (rs_t * rs, rs_resource_t * resource)`

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

Parameters

<i>rs</i>	
<i>resource</i>	

Returns

Definition at line 91 of file [rs_util.c](#).

5.27.2.3 `void rs_check_for_eor_reached (rs_resource_t * resource)`

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

Parameters

<i>resource</i>	
-----------------	--

Definition at line 79 of file [rs_util.c](#).

5.27.2.4 `void rs_create_cluster_chain (rs_t * rs, rs_cluster_t prev_cluster, rs_cluster_t next_cluster)`

Create a chain between two clusters.

Parameters

<i>rs</i>	
<i>prev_cluster</i>	
<i>next_cluster</i>	

Definition at line 67 of file [rs_util.c](#).

5.27.2.5 `void rs_format_cluster (rs_t * rs, rs_cluster_t cluster)`

Format a given cluster.

Parameters

<i>rs</i>	
-----------	--

<i>cluster</i>	
----------------	--

Definition at line 55 of file [rs_util.c](#).

5.27.2.6 uint8_t rs_format_clusters_chain (rs_t * rs, rs_cluster_t cluster)

Format a chain of clusters.

Parameters

<i>rs</i>	
<i>cluster</i>	

Returns

Definition at line 196 of file [rs_util.c](#).

5.27.2.7 void rs_format_resource_descriptor (rs_t * rs, rs_resource_descriptor_t resource_descriptor)

Free a resource description.

Parameters

<i>rs</i>	
<i>resource_↔ descriptor</i>	

Definition at line 153 of file [rs_util.c](#).

5.27.2.8 void rs_format_resource_clusters (rs_t * rs, rs_resource_t * resource)

Free resource cluster.

Parameters

<i>rs</i>	
<i>resource</i>	

Definition at line 190 of file [rs_util.c](#).

5.27.2.9 void rs_free_cluster (rs_t * rs, rs_cluster_t cluster)

Free a given cluster.

Parameters

<i>rs</i>	
<i>cluster</i>	

Definition at line 62 of file [rs_util.c](#).

5.27.2.10 void rs_free_resource_descriptor (rs_t * rs, rs_resource_descriptor_t resource_descriptor)

Close a single resources.

Parameters

<i>rs</i>	
<i>resource_↔ descriptor</i>	

Definition at line 181 of file [rs_util.c](#).

5.27.2.11 `void _rs_free_resource_descriptors (rs_t * rs)`

Close all resources.

Parameters

<i>rs</i>	
-----------	--

Definition at line 174 of file [rs_util.c](#).

5.27.2.12 uint8_t rs_has_invalid_attributes (rs_t * rs)

Calculates and evaluate the rs attributes.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 211 of file [rs_util.c](#).

5.27.2.13 uint8_t rs_is_driver_monted (rs_driver_t driver)

Test if given driver is mouted.

Parameters

<i>driver</i>	
---------------	--

Returns

Definition at line 162 of file [rs_util.c](#).

5.27.2.14 uint8_t rs_is_eor_reached (rs_resource_t * resource)

Test the end-of-resource flag.

Parameters

<i>resource</i>	
-----------------	--

Returns

Definition at line 87 of file [rs_util.c](#).

5.27.2.15 uint8_t rs_is_free_cluster (rs_t * rs, rs_cluster_t cluster)

Test if the given cluster is free.

Parameters

<i>rs</i>	
<i>cluster</i>	

Returns

Definition at line 50 of file [rs_util.c](#).

5.27.2.16 uint8_t rs_move_current_position_ahead (rs_t * rs, rs_resource_t * resource, rs_seek_int_t offset)

Move the current position ahead 'offset' bytes.

Parameters

<i>rs</i>	
<i>resource</i>	
<i>offset</i>	

Returns

Definition at line 112 of file [rs_util.c](#).

5.27.2.17 `uint8_t rs_move_current_position_back (rs_t * rs, rs_resource_t * resource, rs_seek_int_t offset)`

Move the current position back 'offset' bytes.

Parameters

<i>rs</i>	
<i>resource</i>	
<i>offset</i>	

Returns

Definition at line 131 of file [rs_util.c](#).

5.27.2.18 `void _rs_read_rs_from_disc (rs_driver_t driver, rs_t * rs)`

Read a resource system table from disc.

Parameters

<i>driver</i>	
<i>rs</i>	

Definition at line 26 of file [rs_util.c](#).

5.27.2.19 `void _rs_set_driver_mouted (rs_driver_t driver, uint8_t is)`

Set/clear given driver as mouted.

Parameters

<i>driver</i>	
<i>is</i>	

Definition at line 166 of file [rs_util.c](#).

5.27.2.20 `void _rs_write_rs_to_disc (rs_driver_t driver, rs_t * rs)`

SDCC - PIC resource system.

[rs_util.h](#)

Util lib for rs

Author

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

Parameters

<i>driver</i>	
<i>rs</i>	

Definition at line 16 of file [rs_util.c](#).

5.28 rs_util.c

```

00001
00011 #ifndef __SDCC_RS_UTIL_C__
00012 #define __SDCC_RS_UTIL_C__ 1
00013
00014 #include "rs_util.h"
00015
00016 void _rs_write_rs_to_disc(rs_driver_t driver,
    rs_t *rs) {
00017     uint8_t i;
00018     uint8_t *rsp;
00019     rs_memory_address_t address = RS_FIRST_ADDRESS_OF_MEMORY;
00020     rsp = (uint8_t *) rs;
00021     for (i = 0; i < sizeof(rs_t); i++) {
00022         _rs_io_write(driver, address++, *(rsp++));
00023     }
00024 }
00025
00026 void _rs_read_rs_from_disc(rs_driver_t driver,
    rs_t *rs) {
00027     uint8_t i;
00028     uint8_t *rsp;
00029     rs_memory_address_t address = RS_FIRST_ADDRESS_OF_MEMORY;
00030     rsp = (uint8_t *) rs;
00031     for (i = 0; i < sizeof(rs_t); i++) {
00032         *(rsp++) = _rs_io_read(driver, address++);
00033     }
00034 }
00035
00036 rs_memory_address_t _rs_alloc_cluster(rs_t *rs) {
00037     rs_memory_address_t address;
00038     uint8_t i;
00039     address = rs->cluster_table_address;
00040     for (i = 0; i < rs->cluster_count; i++) {
00041         if (_rs_is_free_cluster(rs, (rs_cluster_t) i)) {
00042             _rs_decrease_free_clusters(rs, 1);
00043             return address;
00044         }
00045         address += rs->sizeof_cluster;
00046     }
00047     return RS_NULL_CLUSTER_ADDRESS;
00048 }
00049
00050 uint8_t _rs_is_free_cluster(rs_t *rs, rs_cluster_t cluster) {
00051     return (cluster == _rs_prev_cluster_by_cluster(rs, cluster)) \
00052         && (cluster == _rs_next_cluster_by_cluster(rs, cluster));
00053 }
00054
00055 void _rs_format_cluster(rs_t *rs, rs_cluster_t cluster) {
00056     rs_memory_address_t address;
00057     address = _rs_cluster_to_address(rs, cluster);
00058     _rs_io_write(rs->driver, CLUSTER_ADDRESS_TO_NEXT(address),
    cluster);
00059     _rs_io_write(rs->driver, CLUSTER_ADDRESS_TO_PREV(address),
    cluster);
00060 }
00061
00062 void _rs_free_cluster(rs_t *rs, rs_cluster_t cluster) {
00063     _rs_format_cluster(rs, cluster);
00064     _rs_increase_free_clusters(rs, 1);
00065 }
00066
00067 void _rs_create_cluster_chain(rs_t *rs, rs_cluster_t prev_cluster,
    rs_cluster_t next_cluster) {
00068     rs_memory_address_t address;
00069     if (prev_cluster != RS_INEXISTENT_CLUSTER) {
00070         address = _rs_cluster_to_address(rs, prev_cluster);
00071         _rs_io_write(rs->driver, CLUSTER_ADDRESS_TO_NEXT(address),
    (uint8_t) next_cluster);
00072     }
00073     if (next_cluster != RS_INEXISTENT_CLUSTER) {
00074         address = _rs_cluster_to_address(rs, next_cluster);
00075         _rs_io_write(rs->driver, CLUSTER_ADDRESS_TO_PREV(address),
    (uint8_t) prev_cluster);
00076     }

```

```

00077 }
00078
00079 void _rs_check_for_eor_reached(rs_resource_t *resource) {
00080     if (resource->current_position >= resource->size) {
00081         resource->flags |= RS_RESOURCE_FLAG_BIT_EOR_REACHED;
00082     } else {
00083         resource->flags &= ~RS_RESOURCE_FLAG_BIT_EOR_REACHED;
00084     }
00085 }
00086
00087 uint8_t _rs_is_eor_reached(rs_resource_t *resource) {
00088     return resource->flags & RS_RESOURCE_FLAG_BIT_EOR_REACHED;
00089 }
00090
00091 uint8_t _rs_check_for_availability(rs_t *rs,
rs_resource_t *resource) {
00092     rs_memory_address_t address;
00093     rs_cluster_t cluster;
00094     _rs_check_for_eor_reached(resource);
00095     if (resource->cluster_offset >= rs->sizeof_cluster) {
00096         if (rs_eor(resource)) {
00097             address = _rs_alloc_cluster(rs);
00098             if (address == RS_NULL_CLUSTER_ADDRESS) {
00099                 return 0;
00100             }
00101             cluster = _rs_address_to_cluster(rs, address);
00102             _rs_create_cluster_chain(rs, resource->
current_cluster, cluster);
00103             resource->current_cluster = cluster;
00104         } else {
00105             resource->current_cluster =
_rs_next_cluster_by_cluster(rs, resource->
current_cluster);
00106         }
00107         resource->cluster_offset = rs->sizeof_cluster_control;
00108     }
00109     return 1;
00110 }
00111
00112 uint8_t _rs_move_current_position_ahead(rs_t *rs,
rs_resource_t *resource, rs_seek_int_t offset) {
00113     uint8_t until_the_end;
00114     uint8_t how_many_clustes_ahead;
00115     uint8_t i;
00116     resource->current_position += offset;
00117     until_the_end = (rs->sizeof_cluster - resource->cluster_offset);
00118     if (offset <= until_the_end) {
00119         resource->cluster_offset += offset;
00120         return 1;
00121     }
00122     offset -= until_the_end;
00123     how_many_clustes_ahead = (offset / rs->sizeof_cluster_data) + 1;
00124     resource->cluster_offset = (offset % rs->sizeof_cluster_data) + rs->
sizeof_cluster_control;
00125     for (i = 0; i < how_many_clustes_ahead; i++) {
00126         resource->current_cluster = _rs_next_cluster_by_cluster(
rs, resource->current_cluster);
00127     }
00128     return 1;
00129 }
00130
00131 uint8_t _rs_move_current_position_back(rs_t *rs,
rs_resource_t *resource, rs_seek_int_t offset) {
00132     uint8_t until_the_begin;
00133     uint8_t how_many_clustes_back;
00134     uint8_t i;
00135     resource->current_position -= offset;
00136     until_the_begin = (resource->cluster_offset - rs->
sizeof_cluster_control);
00137     if (offset <= until_the_begin) {
00138         resource->cluster_offset -= offset;
00139         return 1;
00140     }
00141     offset -= until_the_begin;
00142     how_many_clustes_back = (offset / rs->sizeof_cluster_data);
00143     if ((offset % rs->sizeof_cluster_data) != 0) {
00144         how_many_clustes_back++;
00145     }
00146     resource->cluster_offset = rs->sizeof_cluster - (offset % rs->
sizeof_cluster_data);
00147     for (i = 0; i < how_many_clustes_back; i++) {
00148         resource->current_cluster = _rs_prev_cluster_by_cluster(
rs, resource->current_cluster);
00149     }
00150     return 1;
00151 }
00152

```

```

00153 void _rs_format_resource_descriptor(rs_t *rs,
00154     rs_resource_descriptor_t resource_descriptor) {
00155     int i;
00156     rs_memory_address_t address;
00157     address = _rs_resource_descriptor_to_address(rs, resource_descriptor)
00158 ;
00159     for (i = 0; i < rs->sizeof_resource_descriptor; i++) {
00160         _rs_io_write(rs->driver, address + i, 0x00);
00161     }
00162 }
00163 uint8_t _rs_is_driver_mouted(rs_driver_t driver) {
00164     return rs_global_flags.driver_mouted & (1 << driver);
00165 }
00166 void _rs_set_driver_mouted(rs_driver_t driver, uint8_t is) {
00167     if (is) {
00168         rs_global_flags.driver_mouted |= (1 << driver);
00169     } else {
00170         rs_global_flags.driver_mouted &= ~(1 << driver);
00171     }
00172 }
00173 }
00174 void _rs_free_resource_descriptors(rs_t *rs) {
00175     uint8_t i;
00176     for (i = 0; i < rs->resource_descriptor_count; i++) {
00177         _rs_free_resource_descriptor(rs, i);
00178     }
00179 }
00180 }
00181 void _rs_free_resource_descriptor(rs_t *rs,
00182     rs_resource_descriptor_t resource_descriptor) {
00183     rs_memory_address_t address;
00184     uint8_t flags;
00185     address = _rs_resource_descriptor_to_address(rs, resource_descriptor)
00186 ;
00187     flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(address));
00188     flags &= ~(RS_RESOURCE_FLAG_BIT_OPENED |
00189 RS_RESOURCE_FLAG_BIT_READ_ONLY);
00190     _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(address), flags);
00191 }
00192 void _rs_format_resource_clusters(rs_t *rs,
00193     rs_resource_t *resource) {
00194     uint8_t freed_clusters;
00195     freed_clusters = _rs_format_clusters_chain(rs, resource->
00196 first_cluster);
00197     _rs_increase_free_clusters(rs, freed_clusters);
00198 }
00199 }
00200 uint8_t _rs_format_clusters_chain(rs_t *rs,
00201     rs_cluster_t cluster) {
00202     rs_cluster_t next_cluster;
00203     uint8_t formatted_clusters = 0;
00204     do {
00205         next_cluster = _rs_next_cluster_by_cluster(rs, cluster);
00206         _rs_format_cluster(rs, cluster);
00207         formatted_clusters++;
00208         if (next_cluster == RS_INEXISTENT_CLUSTER || next_cluster == cluster) {
00209             break;
00210         }
00211         cluster = next_cluster;
00212     } while (1);
00213     return formatted_clusters;
00214 }
00215 }
00216 uint8_t _rs_has_invalid_attributes(rs_t *rs) {
00217     if (rs->sizeof_resource_descriptor_table != (rs->
00218 sizeof_resource_descriptor * rs->
00219 resource_descriptor_count)) {
00220         return 1;
00221     }
00222     if (rs->sizeof_cluster_table != (rs->sizeof_cluster * rs->
00223 cluster_count)) {
00224         return 2;
00225     }
00226     if (rs->sizeof_cluster != (rs->sizeof_cluster_control + rs->
00227 sizeof_cluster_data)) {
00228         return 3;
00229     }
00230     if (rs->memory_size != rs->sizeof_cluster_table + rs->
00231 cluster_table_address) {
00232         return 4;
00233     }
00234     return 0;
00235 }
00236 }

```

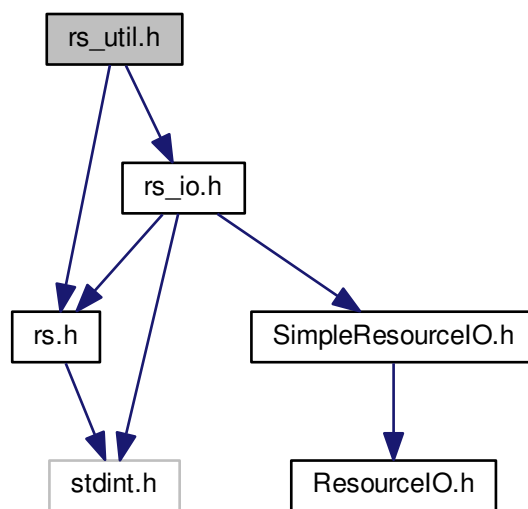
```
00227 #endif // __SDCC_RS_UTIL_C__
```

5.29 rs_util.h File Reference

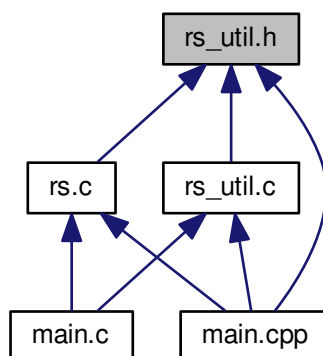
```
#include "rs_io.h"
```

```
#include "rs.h"
```

Include dependency graph for rs_util.h:



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



Macros

- `#define _rs_resource_code_to_resource_descriptor(resource_code) (rs_resource_descriptor_t)(resource↔_code)`
- `#define _rs_resource_descriptor_to_resource_code(resource_descriptor) (rs_resource_code_t)(resource↔_descriptor)`
- `#define _rs_cluster_to_address(rs, cluster) (rs_memory_address_t)(rs->cluster_table_address + (cluster * rs->sizeof_cluster))`
- `#define _rs_address_to_cluster(rs, address) (rs_cluster_t)((address - rs->cluster_table_address) / rs->sizeof_cluster)`
- `#define _rs_resource_descriptor_to_address(rs, resource_descriptor) (rs_memory_address_t)((resource↔_descriptor * rs->sizeof_resource_descriptor) + rs->resource_descriptor_table_address)`
- `#define _rs_address_to_resource_descriptor(rs, address) (rs_resource_descriptor_t)((address - rs->resource_descriptor_table_address) / rs->sizeof_resource_descriptor)`
- `#define _rs_decrease_free_clusters(rs, n)`
- `#define _rs_increase_free_clusters(rs, n)`
- `#define _rs_prev_cluster_by_cluster(rs, cluster) _rs_prev_cluster_by_cluster_address(rs, _rs_cluster_to_↔address(rs, cluster))`
- `#define _rs_next_cluster_by_cluster(rs, cluster) _rs_next_cluster_by_cluster_address(rs, _rs_cluster_to_↔address(rs, cluster))`
- `#define _rs_prev_cluster_by_cluster_address(rs, address) (rs_cluster_t)(_rs_io_read(rs->driver, CLUSTE↔R_ADDRESS_TO_PREV(address)))`
- `#define _rs_next_cluster_by_cluster_address(rs, address) (rs_cluster_t)(_rs_io_read(rs->driver, CLUSTE↔R_ADDRESS_TO_NEXT(address)))`

Functions

- `void _rs_write_rs_to_disc (rs_driver_t driver, rs_t *rs)`
- `void _rs_read_rs_from_disc (rs_driver_t driver, rs_t *rs)`
- `rs_memory_address_t _rs_alloc_cluster (rs_t *rs)`
- `uint8_t _rs_is_free_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_format_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_free_cluster (rs_t *rs, rs_cluster_t cluster)`
- `void _rs_create_cluster_chain (rs_t *rs, rs_cluster_t prev_cluster, rs_cluster_t next_cluster)`
- `void _rs_check_for_eor_reached (rs_resource_t *resource)`
- `uint8_t _rs_is_eor_reached (rs_resource_t *resource)`
- `uint8_t _rs_check_for_availability (rs_t *rs, rs_resource_t *resource)`
- `uint8_t _rs_move_current_position_ahead (rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset)`
- `uint8_t _rs_move_current_position_back (rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset)`
- `void _rs_format_resource_descriptor (rs_t *rs, rs_resource_descriptor_t resource_descriptor)`
- `uint8_t _rs_is_driver_monted (rs_driver_t driver)`
- `void _rs_set_driver_monted (rs_driver_t driver, uint8_t is)`
- `void _rs_free_resource_descriptors (rs_t *rs)`
- `void _rs_free_resource_descriptor (rs_t *rs, rs_resource_descriptor_t resource_descriptor)`
- `void _rs_format_resource_clusters (rs_t *rs, rs_resource_t *resource)`
- `uint8_t _rs_format_clusters_chain (rs_t *rs, rs_cluster_t cluster)`
- `uint8_t _rs_has_invalid_attributes (rs_t *rs)`

5.29.1 Macro Definition Documentation

5.29.1.1 `#define _rs_address_to_cluster(rs, address) (rs_cluster_t)((address - rs->cluster_table_address) / rs->sizeof_cluster)`

Convert address to cluster.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 101 of file [rs_util.h](#).

5.29.1.2 `#define _rs_address_to_resource_descriptor(rs, address) (rs_resource_descriptor_t)((address - rs->resource_descriptor_table_address) / rs->sizeof_resource_descriptor)`

Convert address to rd.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 115 of file [rs_util.h](#).

5.29.1.3 `#define _rs_cluster_to_address(rs, cluster) (rs_memory_address_t)(rs->cluster_table_address + (cluster * rs->sizeof_cluster))`

Convert cluster to address.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 94 of file [rs_util.h](#).

5.29.1.4 `#define _rs_decrease_free_clusters(rs, n)`

Value:

```
{ \
                                rs->free_clusters -= n; \
                                _rs_write_rs_to_disc
(rs->driver, rs); \
}
```

Decrease free cluster.

Parameters

<i>rs</i>	
<i>resource</i>	

Definition at line 207 of file [rs_util.h](#).

5.29.1.5 `#define _rs_increase_free_clusters(rs, n)`

Value:

```
{ \
                                rs->free_clusters += n; \
                                _rs_write_rs_to_disc
(rs->driver, rs); \
}
```

Increase free cluster.

Parameters

<i>rs</i>	
<i>resource</i>	

Definition at line 218 of file [rs_util.h](#).

5.29.1.6 `#define _rs_next_cluster_by_cluster(rs, cluster) _rs_next_cluster_by_cluster_address(rs, _rs_cluster_to_address(rs, cluster))`

Get the next cluster by a cluster.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 254 of file [rs_util.h](#).

```
5.29.1.7 #define _rs_next_cluster_by_cluster_address( rs, address ) (rs_cluster_t)(_rs_io_read(rs->driver,
    CLUSTER_ADDRESS_TO_NEXT(address)))
```

Get the next cluster by a cluster address.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 270 of file [rs_util.h](#).

```
5.29.1.8 #define _rs_prev_cluster_by_cluster( rs, cluster ) _rs_prev_cluster_by_cluster_address(rs,
    _rs_cluster_to_address(rs, cluster))
```

Get the previous cluster by a cluster.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 246 of file [rs_util.h](#).

```
5.29.1.9 #define _rs_prev_cluster_by_cluster_address( rs, address ) (rs_cluster_t)(_rs_io_read(rs->driver,
    CLUSTER_ADDRESS_TO_PREV(address)))
```

Get the previous cluster by a cluster address.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 262 of file [rs_util.h](#).

```
5.29.1.10 #define _rs_resource_code_to_resource_descriptor( resource_code ) (rs_resource_descriptor_t)(resource_↔
    code)
```

Convert resource code to rd.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 80 of file [rs_util.h](#).

5.29.1.11 `#define _rs_resource_descriptor_to_address(rs, resource_descriptor) (rs_memory_address_t)((resource_↵
descriptor * rs->sizeof_resource_descriptor) + rs->resource_descriptor_table_address)`

Convert rd to address.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 108 of file [rs_util.h](#).

5.29.1.12 `#define _rs_resource_descriptor_to_resource_code(resource_descriptor) (rs_resource_code_t)(resource_↵
descriptor)`

Convert rd to resource code.

Parameters

<i>resource</i>	
-----------------	--

Definition at line 87 of file [rs_util.h](#).

5.29.2 Function Documentation

5.29.2.1 `rs_memory_address_t rs_alloc_cluster (rs_t * rs)`

Allocate a free cluster from disc if any.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 36 of file [rs_util.c](#).

5.29.2.2 `uint8_t rs_check_for_availability (rs_t * rs, rs_resource_t * resource)`

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

Parameters

<i>rs</i>	
<i>resource</i>	

Returns

Definition at line 91 of file [rs_util.c](#).

5.29.2.3 `void rs_check_for_eor_reached (rs_resource_t * resource)`

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

Parameters

<i>resource</i>	
-----------------	--

Definition at line 79 of file [rs_util.c](#).

5.29.2.4 void `_rs_create_cluster_chain` (`rs_t * rs`, `rs_cluster_t prev_cluster`, `rs_cluster_t next_cluster`)

Create a chain between two clusters.

Parameters

<i>rs</i>	
<i>prev_cluster</i>	
<i>next_cluster</i>	

Definition at line 67 of file [rs_util.c](#).

5.29.2.5 void `_rs_format_cluster` (`rs_t * rs`, `rs_cluster_t cluster`)

Format a given cluster.

Parameters

<i>rs</i>	
<i>cluster</i>	

Definition at line 55 of file [rs_util.c](#).

5.29.2.6 uint8_t `_rs_format_clusters_chain` (`rs_t * rs`, `rs_cluster_t cluster`)

Format a chain of clusters.

Parameters

<i>rs</i>	
<i>cluster</i>	

Returns

Definition at line 196 of file [rs_util.c](#).

5.29.2.7 void `_rs_format_resorce_descriptor` (`rs_t * rs`, `rs_resource_descriptor_t resource_descriptor`)

Free a resource description.

Parameters

<i>rs</i>	
<i>resource_↔ descriptor</i>	

Definition at line 153 of file [rs_util.c](#).

5.29.2.8 void `_rs_format_resource_clusters` (`rs_t * rs`, `rs_resource_t * resource`)

Free resource cluster.

Parameters

<i>rs</i>	
<i>resource</i>	

Definition at line 190 of file [rs_util.c](#).

5.29.2.9 void `_rs_free_cluster` (*rs_t* * *rs*, *rs_cluster_t* *cluster*)

Free a given cluster.

Parameters

<i>rs</i>	
<i>cluster</i>	

Definition at line 62 of file [rs_util.c](#).

5.29.2.10 void `_rs_free_resource_descriptor` (*rs_t* * *rs*, *rs_resource_descriptor_t* *resource_descriptor*)

Close a single resources.

Parameters

<i>rs</i>	
<i>resource_↔ descriptor</i>	

Definition at line 181 of file [rs_util.c](#).

5.29.2.11 void `_rs_free_resource_descriptors` (*rs_t* * *rs*)

Close all resources.

Parameters

<i>rs</i>	
-----------	--

Definition at line 174 of file [rs_util.c](#).

5.29.2.12 uint8_t `_rs_has_invalid_attributes` (*rs_t* * *rs*)

Calculates and evaluate the rs attributes.

Parameters

<i>rs</i>	
-----------	--

Returns

Definition at line 211 of file [rs_util.c](#).

5.29.2.13 uint8_t `_rs_is_driver_mouted` (*rs_driver_t* *driver*)

Test if given driver is mouted.

Parameters

<i>driver</i>	
---------------	--

Returns

Definition at line 162 of file [rs_util.c](#).

5.29.2.14 `uint8_t _rs_is_eor_reached (rs_resource_t * resource)`

Test the end-of-resource flag.

Parameters

<i>resource</i>	
-----------------	--

Returns

Definition at line 87 of file [rs_util.c](#).

5.29.2.15 `uint8_t rs_is_free_cluster (rs_t * rs, rs_cluster_t cluster)`

Test if the given cluster is free.

Parameters

<i>rs</i>	
<i>cluster</i>	

Returns

Definition at line 50 of file [rs_util.c](#).

5.29.2.16 `uint8_t rs_move_current_position_ahead (rs_t * rs, rs_resource_t * resource, rs_seek_int_t offset)`

Move the current position ahead 'offset' bytes.

Parameters

<i>rs</i>	
<i>resource</i>	
<i>offset</i>	

Returns

Definition at line 112 of file [rs_util.c](#).

5.29.2.17 `uint8_t rs_move_current_position_back (rs_t * rs, rs_resource_t * resource, rs_seek_int_t offset)`

Move the current position back 'offset' bytes.

Parameters

<i>rs</i>	
<i>resource</i>	
<i>offset</i>	

Returns

Definition at line 131 of file [rs_util.c](#).

5.29.2.18 `void _rs_read_rs_from_disc (rs_driver_t driver, rs_t * rs)`

Read a resource system table from disc.

Parameters

<i>driver</i>	
<i>rs</i>	

Definition at line 26 of file [rs_util.c](#).

5.29.2.19 void `_rs_set_driver_mouted (rs_driver_t driver, uint8_t is)`

Set/clear given driver as mouted.

Parameters

<i>driver</i>	
<i>is</i>	

Definition at line 166 of file [rs_util.c](#).

5.29.2.20 void `_rs_write_rs_to_disc (rs_driver_t driver, rs_t * rs)`

SDCC - PIC resource system.

[rs_util.h](#)

Util lib for rs

Author

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

Parameters

<i>driver</i>	
<i>rs</i>	

Definition at line 16 of file [rs_util.c](#).

5.30 rs_util.h

```

00001
00011 #ifndef __SDCC_RS_UTIL_H__
00012 #define __SDCC_RS_UTIL_H__ 1
00013
00014 #include "rs_io.h"
00015 #include "rs.h"
00016
00023 void _rs_write_rs_to_disc(rs_driver_t driver,
    rs_t *rs);
00024
00031 void _rs_read_rs_from_disc(rs_driver_t driver,
    rs_t *rs);
00032
00039 rs_memory_address_t _rs_alloc_cluster(rs_t *rs);
00040
00048 uint8_t _rs_is_free_cluster(rs_t *rs, rs_cluster_t cluster);
00049
00056 void _rs_format_cluster(rs_t *rs, rs_cluster_t cluster);
00057
00064 void _rs_free_cluster(rs_t *rs, rs_cluster_t cluster);
00065
00073 void _rs_create_cluster_chain(rs_t *rs, rs_cluster_t prev_cluster,
    rs_cluster_t next_cluster);
00074
00080 #define _rs_resource_code_to_resource_descriptor(resource_code)
    (rs_resource_descriptor_t)(resource_code)
00081
00087 #define _rs_resource_descriptor_to_resource_code(resource_descriptor)
    (rs_resource_code_t)(resource_descriptor)
00088
00094 #define _rs_cluster_to_address(rs, cluster)
    (rs_memory_address_t)(rs->cluster_table_address + (cluster * rs->sizeof_cluster))
00095
00101 #define _rs_address_to_cluster(rs, address)
    (rs_cluster_t)((address -
    rs->cluster_table_address) / rs->sizeof_cluster)

```

```

00102
00108 #define _rs_resource_descriptor_to_address(rs, resource_descriptor)
      (rs_memory_address_t)((resource_descriptor * rs->sizeof_resource_descriptor) + rs->resource_descriptor_table_address)
00109
00115 #define _rs_address_to_resource_descriptor(rs, address) (rs_resource_descriptor_t)((address
      - rs->resource_descriptor_table_address) / rs->sizeof_resource_descriptor)
00116
00122 void _rs_check_for_eor_reached(rs_resource_t *resource);
00123
00130 uint8_t _rs_is_eor_reached(rs_resource_t *resource);
00131
00140 uint8_t _rs_check_for_availability(rs_t *rs,
      rs_resource_t *resource);
00141
00150 uint8_t _rs_move_current_position_ahead(rs_t *rs,
      rs_resource_t *resource, rs_seek_int_t offset);
00151
00160 uint8_t _rs_move_current_position_back(rs_t *rs,
      rs_resource_t *resource, rs_seek_int_t offset);
00161
00168 void _rs_format_resource_descriptor(rs_t *rs,
      rs_resource_descriptor_t resource_descriptor);
00169
00176 uint8_t _rs_is_driver_monted(rs_driver_t driver);
00177
00184 void _rs_set_driver_monted(rs_driver_t driver, uint8_t is);
00185
00191 void _rs_free_resource_descriptors(rs_t *rs);
00192
00199 void _rs_free_resource_descriptor(rs_t *rs,
      rs_resource_descriptor_t resource_descriptor);
00200
00207 #define _rs_decrease_free_clusters(rs, n) { \
00208     rs->free_clusters -= n; \
00209     _rs_write_rs_to_disc(rs->driver, rs); \
00210 }
00211
00218 #define _rs_increase_free_clusters(rs, n) { \
00219     rs->free_clusters += n; \
00220     _rs_write_rs_to_disc(rs->driver, rs); \
00221 }
00222
00229 void _rs_format_resource_clusters(rs_t *rs,
      rs_resource_t *resource);
00230
00238 uint8_t _rs_format_clusters_chain(rs_t *rs,
      rs_cluster_t cluster);
00239
00246 #define _rs_prev_cluster_by_cluster(rs, cluster) _rs_prev_cluster_by_cluster_address(rs,
      _rs_cluster_to_address(rs, cluster))
00247
00254 #define _rs_next_cluster_by_cluster(rs, cluster) _rs_next_cluster_by_cluster_address(rs,
      _rs_cluster_to_address(rs, cluster))
00255
00262 #define _rs_prev_cluster_by_cluster_address(rs, address) (rs_cluster_t)(_rs_io_read(rs->driver,
      CLUSTER_ADDRESS_TO_PREV(address)))
00263
00270 #define _rs_next_cluster_by_cluster_address(rs, address) (rs_cluster_t)(_rs_io_read(rs->driver,
      CLUSTER_ADDRESS_TO_NEXT(address)))
00271
00278 uint8_t _rs_has_invalid_attributes(rs_t *rs);
00279
00280 #endif // __SDCC_RS_UTIL_H__

```

5.31 rs_util_spec.h File Reference

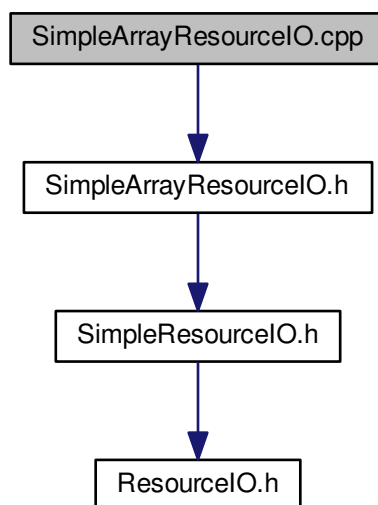
5.32 rs_util_spec.h

```
00001
```

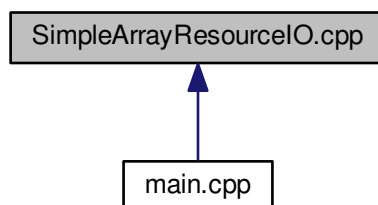
5.33 SimpleArrayResourceIO.cpp File Reference

```
#include "SimpleArrayResourceIO.h"
```

Include dependency graph for SimpleArrayResourceIO.cpp:



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



Macros

- `#define __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__ 1`

5.33.1 Macro Definition Documentation

5.33.1.1 `#define __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__ 1`

Arduino - A simple resource implementation.

[SimpleArrayResourceIO.cpp](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleArrayResourceIO.cpp](#).

5.34 SimpleArrayResourceIO.cpp

```

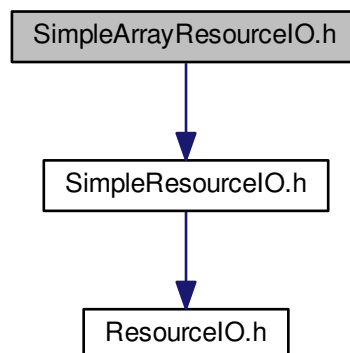
00001
00011 #ifndef __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__
00012 #define __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__ 1
00013
00014 #include "SimpleArrayResourceIO.h"
00015
00016 SimpleArrayResourceIO::SimpleArrayResourceIO(unsigned char*
array, unsigned int size) : SimpleResourceIO(), array(array), size(size) {
00017 }
00018
00019 int SimpleArrayResourceIO::readBytes(unsigned int address, unsigned char*
buf, int len) {
00020     unsigned int available = (size - address);
00021     if (available < 1) {
00022         return -1;
00023     }
00024     len = (len > available) ? available : len;
00025     for (int i = 0; i < len; i++) {
00026         buf[i] = array[address + i];
00027     }
00028     return len;
00029 }
00030
00031 void SimpleArrayResourceIO::writeBytes(unsigned int address, unsigned char
* buf, int len) {
00032     for (int i = 0; i < len && (address + i) < size; i++) {
00033         array[address + i] = buf[i];
00034     }
00035 }
00036
00037 #endif /* __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__ */
00038

```

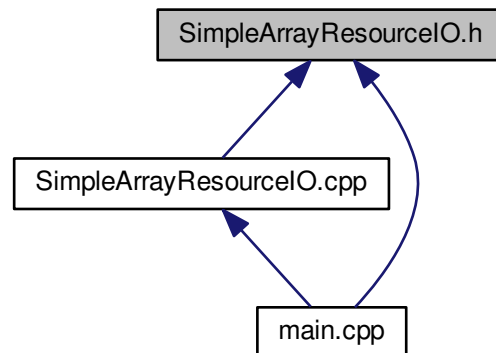
5.35 SimpleArrayResourceIO.h File Reference

```
#include <SimpleResourceIO.h>
```

Include dependency graph for SimpleArrayResourceIO.h:



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



Classes

- class [SimpleArrayResourceIO](#)

5.36 SimpleArrayResourceIO.h

```

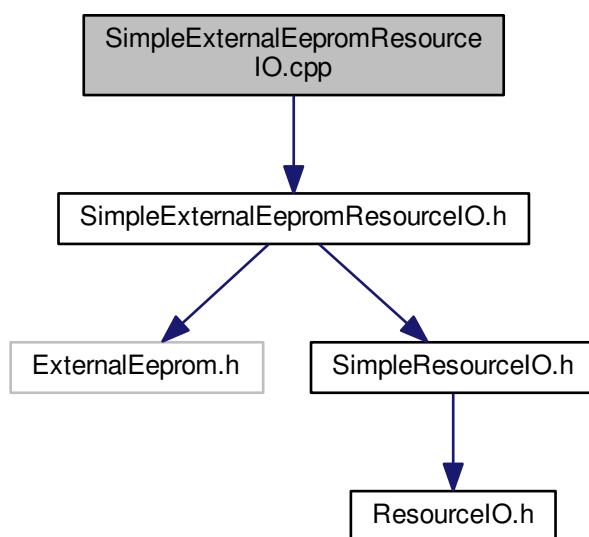
00001
00011 #ifndef __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__
00012 #define __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__ 1
00013
00014 #include <SimpleResourceIO.h>
00015
00016 class SimpleArrayResourceIO : public SimpleResourceIO {
00017 private:
00018     unsigned char* array;
00019     unsigned int size;
00020 public:
00021
00022     SimpleArrayResourceIO(unsigned char* array, unsigned int size);
00023
00024 protected:
00025
00026     virtual int readBytes(unsigned int address, unsigned char* buf, int len);
00027
00028     virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
00029 };
00030
00031 #endif /* __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__ */
00032

```

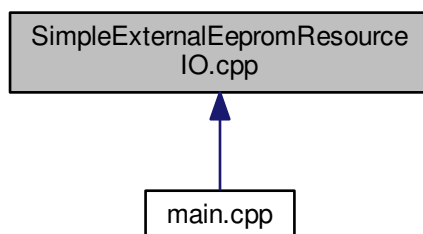
5.37 SimpleExternalEepromResourceIO.cpp File Reference

```
#include "SimpleExternalEepromResourceIO.h"
```

Include dependency graph for SimpleExternalEepromResourceIO.cpp:



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



Macros

- `#define __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__ 1`

5.37.1 Macro Definition Documentation

5.37.1.1 `#define __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__ 1`

Arduino - A simple resource implementation.

[SimpleExternalEepromResourceIO.cpp](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleExternalEepromResourceIO.cpp](#).

5.38 SimpleExternalEepromResourceIO.cpp

```

00001
00011 #ifndef __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__
00012 #define __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__ 1
00013
00014 #include "SimpleExternalEepromResourceIO.h"
00015
00016 SimpleExternalEepromResourceIO::SimpleExternalEepromResourceIO
00017 (ExternalEeprom* externalEeprom) : SimpleResourceIO(), externalEeprom(externalEeprom) {
00018 }
00019
00019 int SimpleExternalEepromResourceIO::readBytes(unsigned int address
00020 , unsigned char* buf, int len) {
00021     return externalEeprom->readBytes(address, buf, len);
00022 }
00023
00023 void SimpleExternalEepromResourceIO::writeBytes(unsigned int
00024 address, unsigned char* buf, int len) {
00025     externalEeprom->writeBytes(address, buf, len);
00026 }
00027
00027 #endif /* __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__ */
00028

```

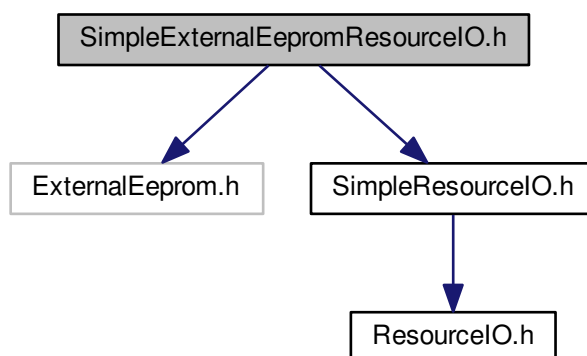
5.39 SimpleExternalEepromResourceIO.h File Reference

```

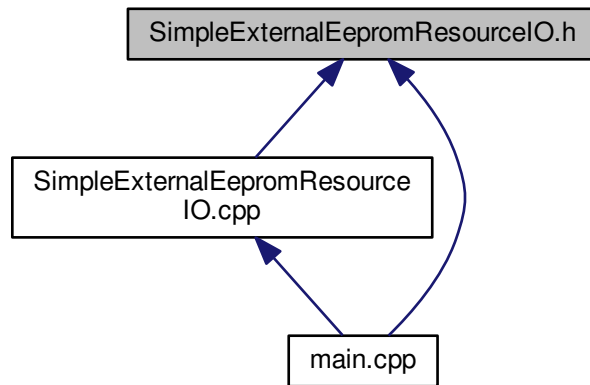
#include <ExternalEeprom.h>
#include <SimpleResourceIO.h>

```

Include dependency graph for SimpleExternalEepromResourceIO.h:



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



Classes

- class [SimpleExternalEepromResourceIO](#)

5.40 SimpleExternalEepromResourceIO.h

```

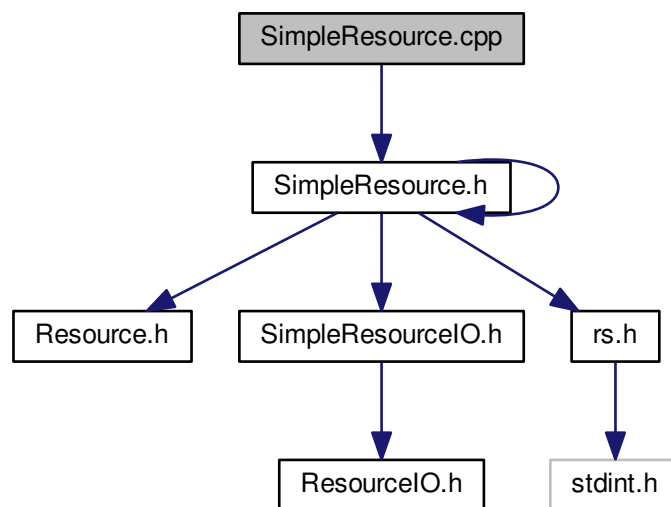
00001
00011 #ifndef __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__
00012 #define __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__ 1
00013
00014 #include <ExternalEeprom.h>
00015 #include <SimpleResourceIO.h>
00016
00017 class SimpleExternalEepromResourceIO : public
SimpleResourceIO {
00018 private:
00019     ExternalEeprom* externalEeprom;
00020 public:
00021
00022     SimpleExternalEepromResourceIO(ExternalEeprom* externalEeprom);
00023
00024 protected:
00025
00026     virtual int readBytes(unsigned int address, unsigned char* buf, int len);
00027
00028     virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
00029 };
00030
00031 #endif /* __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__ */
00032

```

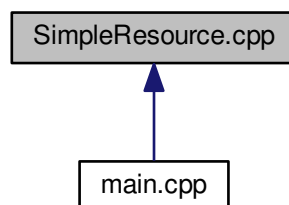
5.41 SimpleResource.cpp File Reference

```
#include "SimpleResource.h"
```

Include dependency graph for SimpleResource.cpp:



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



Macros

- `#define __ARDUINO_SIMPLE_RESOURCE_CPP__ 1`

5.41.1 Macro Definition Documentation

5.41.1.1 `#define __ARDUINO_SIMPLE_RESOURCE_CPP__ 1`

Arduino - A simple resource implementation.

[SimpleResource.cpp](#)

This is the [Resource](#) representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleResource.cpp](#).

5.42 SimpleResource.cpp

```

00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_CPP__
00012 #define __ARDUINO_SIMPLE_RESOURCE_CPP__ 1
00013
00014 #include "SimpleResource.h"
00015
00016 SimpleResource::SimpleResource(rs_resource_code_t code,
    rs_t* rs) : code(code), rs(rs) {
00017     lastOperationResult = OPERATION_SUCCESS;
00018 }
00019
00020 bool SimpleResource::open(OpenOptions options) {
00021     lastOperationResult = (ResourceOperationResult)
    rs_open(rs, code, &resource, (rs_open_resource_options_t)
    options);
00022     return (lastOperationResult == OPERATION_SUCCESS);
00023 }
00024
00025 bool SimpleResource::close() {
00026     sync();
00027     lastOperationResult = (ResourceOperationResult)
    rs_close(rs, &resource);
00028     return (lastOperationResult == OPERATION_SUCCESS);
00029 }
00030
00031 void SimpleResource::write(unsigned char b) {
00032     lastOperationResult = (ResourceOperationResult)
    rs_write(rs, &resource, b);
00033 }
00034
00035 void SimpleResource::writeBytes(unsigned char* buf, int count) {
00036     lastOperationResult = OPERATION_SUCCESS;
00037     for (int i = 0; i < count && lastOperationResult ==
    OPERATION_SUCCESS; i++) {
00038         write(buf[i]);
00039     }
00040 }
00041
00042 int SimpleResource::read() {
00043     if (eor()) {
00044         return -1;
00045     }
00046     return rs_read(rs, &resource);
00047 }
00048
00049 int SimpleResource::readBytes(unsigned char* buf, int count) {
00050     int i, c;
00051     if (buf == (unsigned char*) 0) {
00052         return 0;
00053     }
00054     c = read();
00055     if (c == -1) {
00056         return -1;
00057     }
00058     buf[0] = c;
00059     for (i = 1; i < count; i++) {
00060         c = read();
00061         if (c == -1) {
00062             break;
00063         }
00064         buf[i] = c;
00065     }
00066     return i;
00067 }
00068
00069 bool SimpleResource::seek(ResourceSeekOrigin origin, unsigned int
    offset) {
00070     lastOperationResult = (ResourceOperationResult)
    rs_seek(rs, &resource, (rs_seek_origin_t) origin, (
    rs_seek_int_t) offset);
00071     return (lastOperationResult == OPERATION_SUCCESS);
00072 }
00073
00074 bool SimpleResource::truncate() {
00075     lastOperationResult = (ResourceOperationResult)

```

```

    rs_truncate(rs, &resource);
00076     return (lastOperationResult == OPERATION_SUCCESS);
00077 }
00078
00079 void SimpleResource::sync() {
00080     rs_sync(rs, &resource);
00081     SimpleResourceIO::getAssociatedIO(rs->
    driver)->flush();
00082 }
00083
00084 bool SimpleResource::rewind() {
00085     lastOperationResult = (ResourceOperationResult)
    rs_rewind(rs, &resource);
00086     return (lastOperationResult == OPERATION_SUCCESS);
00087 }
00088
00089 void SimpleResource::release() {
00090     sync();
00091     rs_release(rs, &resource);
00092 }
00093
00094 unsigned int SimpleResource::size() {
00095     return (unsigned int) rs_size(&resource);
00096 }
00097
00098 unsigned int SimpleResource::tell() {
00099     return (unsigned int) rs_tell(&resource);
00100 }
00101
00102 bool SimpleResource::eor() {
00103     return (rs_eor(&resource) != 0);
00104 }
00105
00106 bool SimpleResource::error() {
00107     return (rs_error(&resource) != 0);
00108 }
00109
00110 bool SimpleResource::isReadOnly() {
00111     return (rs->flags & RS_RESOURCE_FLAG_BIT_READ_ONLY) != 0;
00112 }
00113
00114 #endif /* __ARDUINO_SIMPLE_RESOURCE_CPP__ */

```

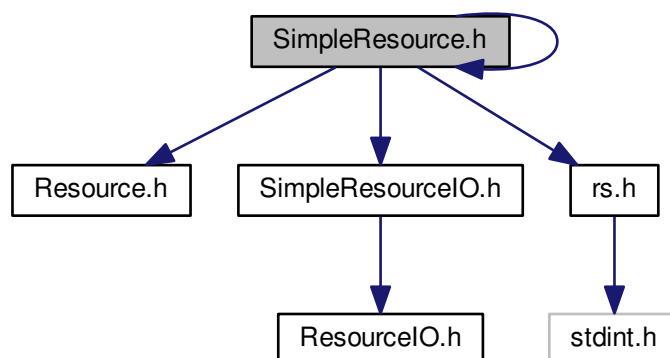
5.43 SimpleResource.h File Reference

```

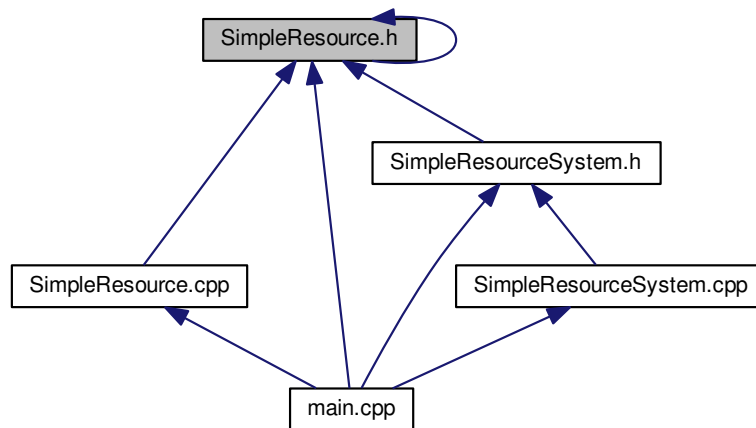
#include <Resource.h>
#include <SimpleResource.h>
#include <SimpleResourceIO.h>
#include <rs.h>

```

Include dependency graph for SimpleResource.h:



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



Classes

- class [SimpleResource](#)

5.44 SimpleResource.h

```

00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_H__
00012 #define __ARDUINO_SIMPLE_RESOURCE_H__ 1
00013
00014 #include <Resource.h>
00015 #include <SimpleResource.h>
00016 #include <SimpleResourceIO.h>
00017 #include <rs.h>
00018
00019 class SimpleResource : public Resource {
00020     rs_resource_code_t code;
00021     rs_resource_t resource;
00022     rs_t* rs;
00023     ResourceOperationResult lastOperationResult;
00024 public:
00025
00026     SimpleResource(rs_resource_code_t code, rs_t* rs);
00027
00028     ResourceOperationResult getLastOperationResult() {
00029         return lastOperationResult;
00030     }
00031
00032     virtual void setCode(int code) {
00033         this->code = (rs_resource_code_t) code;
00034     }
00035
00036     virtual int getCode() {
00037         return (int) this->code;
00038     }
00039
00040     virtual bool open(OpenOptions options);
00041
00042     virtual bool close();
00043
00044     virtual void write(unsigned char b);
00045
00046     virtual void writeBytes(unsigned char* buf, int len);
00047
00048     virtual int read();
00049
00050     virtual int readBytes(unsigned char* buf, int len);
00051

```

```

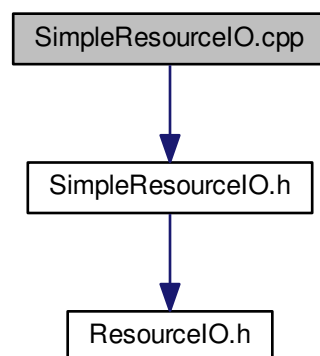
00052     virtual bool seek(ResourceSeekOrigin origin, unsigned int offset);
00053
00054     virtual bool truncate();
00055
00056     virtual void sync();
00057
00058     virtual bool rewind();
00059
00060     virtual void release();
00061
00062     virtual unsigned int size();
00063
00064     virtual unsigned int tell();
00065
00066     virtual bool eor();
00067
00068     virtual bool error();
00069
00070     virtual bool isReadOnly();
00071 };
00072
00073 #endif // __ARDUINO_SIMPLE_RESOURCE_H__

```

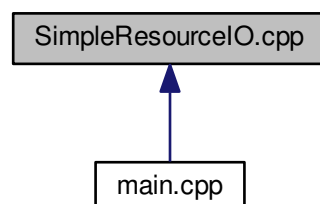
5.45 SimpleResourceIO.cpp File Reference

```
#include "SimpleResourceIO.h"
```

Include dependency graph for SimpleResourceIO.cpp:



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



Macros

- `#define __ARDUINO_SIMPLE_RESOURCE_IO_CPP__ 1`

5.45.1 Macro Definition Documentation

5.45.1.1 `#define __ARDUINO_SIMPLE_RESOURCE_IO_CPP__ 1`

Arduino - A simple resource implementation.

[SimpleResourceIO.cpp](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleResourceIO.cpp](#).

5.46 SimpleResourceIO.cpp

```

00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_IO_CPP__
00012 #define __ARDUINO_SIMPLE_RESOURCE_IO_CPP__ 1
00013
00014 #include "SimpleResourceIO.h"
00015
00016 SimpleResourceIO* SimpleResourceIO::association[
    RESOURCE_IO_DRIVERS_NUM];
00017
00018 SimpleResourceIO* SimpleResourceIO::getAssociatedIO(int
    driver) {
00019     return association[driver];
00020 }
00021
00022 void SimpleResourceIO::associateIO(SimpleResourceIO* io, int
    driver) {
00023     association[driver] = io;
00024 }
00025
00026 bool SimpleResourceIO::open() {
00027     return true;
00028 }
00029
00030 int SimpleResourceIO::read(unsigned int address) {
00031     checkCache(address);
00032     if (validCacheSize < 1) {
00033         return -1;
00034     }
00035     return (int) cache[address - cacheMemoryAddress];
00036 }
00037
00038 void SimpleResourceIO::write(unsigned int address, unsigned char b) {
00039     checkCache(address);
00040     cache[address - cacheMemoryAddress] = b;
00041     wasCacheChanged = true;
00042 }
00043
00044 void SimpleResourceIO::flush() {
00045     if (wasCacheChanged) {
00046         writeBytes(cacheMemoryAddress, cache,
            validCacheSize);
00047     }
00048 }
00049
00050 void SimpleResourceIO::close() {
00051     flush();
00052 }
00053
00054 #endif /* __ARDUINO_SIMPLE_RESOURCE_IO_CPP__ */
00055

```


5.47.1.2 #define RESOURCE_IO_DRIVERS_NUM 5

Definition at line 17 of file [SimpleResourceIO.h](#).

5.48 SimpleResourceIO.h

```

00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_IO_H__
00012 #define __ARDUINO_SIMPLE_RESOURCE_IO_H__ 1
00013
00014 #include <ResourceIO.h>
00015
00016 #define RESOURCE_IO_CACHE_SIZE 8
00017 #define RESOURCE_IO_DRIVERS_NUM 5
00018
00019 class SimpleResourceIO : public ResourceIO {
00020 private:
00021
00022     static SimpleResourceIO* association[
00023         RESOURCE_IO_DRIVERS_NUM];
00024     bool wasCacheChanged, wasCacheInitialized;
00025     unsigned int cacheMemoryAddress;
00026     unsigned char cache[RESOURCE_IO_CACHE_SIZE];
00027     unsigned int cacheMiss, cacheHit;
00028     unsigned int validCacheSize;
00029
00029     void checkCache(unsigned int address) {
00030         if (!wasCacheInitialized || (address < cacheMemoryAddress || address >= (cacheMemoryAddress +
00031             validCacheSize))) {
00032             flush();
00033             validCacheSize = readBytes(address, cache,
00034                 RESOURCE_IO_CACHE_SIZE);
00035             cacheMemoryAddress = address;
00036             wasCacheChanged = false;
00037             wasCacheInitialized = true;
00038             cacheMiss++;
00039         } else {
00040             cacheHit++;
00041         }
00042     }
00043 protected:
00044     SimpleResourceIO() {
00045         cacheMiss = 0;
00046         cacheHit = 0;
00047         cacheMemoryAddress = 0;
00048         wasCacheChanged = false;
00049         wasCacheInitialized = false;
00050         validCacheSize = 0;
00051     }
00052
00053     virtual int readBytes(unsigned int address, unsigned char* buf, int len) {
00054     }
00055
00056     virtual void writeBytes(unsigned int address, unsigned char* buf, int len) {
00057     }
00058
00059 public:
00060
00061     static void associateIO(SimpleResourceIO* io, int driver);
00062
00063     static SimpleResourceIO* getAssociatedIO(int driver);
00064
00065     virtual bool open();
00066
00067     virtual int read(unsigned int address);
00068
00069     virtual void write(unsigned int address, unsigned char b);
00070
00071     virtual void flush();
00072
00073     virtual void close();
00074
00075     unsigned int getCacheHit() {
00076         return cacheHit;
00077     }
00078
00079     unsigned int getCacheMiss() {
00080         return cacheMiss;
00081     }
00082 };
00083

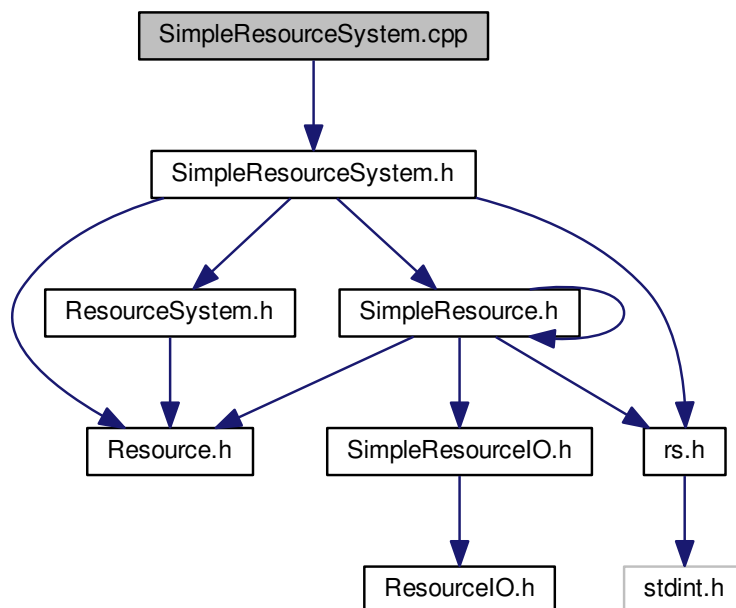
```

```
00084 #endif /* __ARDUINO_SIMPLE_RESOURCE_IO_H__ */
00085
```

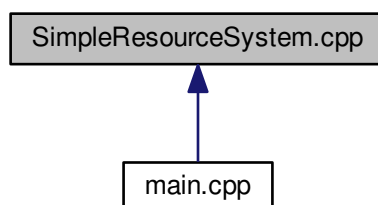
5.49 SimpleResourceSystem.cpp File Reference

```
#include "SimpleResourceSystem.h"
```

Include dependency graph for SimpleResourceSystem.cpp:



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



Macros

- #define `__ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__` 1

5.49.1 Macro Definition Documentation

5.49.1.1 #define __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__ 1

Arduino - A simple resource implementation.

[SimpleResourceSystem.cpp](#)

This is the [Resource](#) system itself.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleResourceSystem.cpp](#).

5.50 SimpleResourceSystem.cpp

```

00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__
00012 #define __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__ 1
00013
00014 #include "SimpleResourceSystem.h"
00015
00016 SimpleResourceSystem::SimpleResourceSystem(int driver) {
00017     lastOperationResult = Resource::OPERATION_SUCCESS;
00018     rs.driver = (rs_driver_t) driver;
00019 }
00020
00021 bool SimpleResourceSystem::mount(MountOptions options) {
00022     lastOperationResult = (Resource::ResourceOperationResult
00023 ) rs_mount(rs.driver, &rs, (rs_mount_options_t) options);
00023     return (lastOperationResult ==
00024 Resource::OPERATION_SUCCESS);
00024 }
00025
00026 bool SimpleResourceSystem::umount() {
00027     SimpleResourceIO::getAssociatedIO(rs.
00028 driver)->flush();
00028     lastOperationResult = (Resource::ResourceOperationResult
00029 ) rs_umount(&rs);
00029     return (lastOperationResult ==
00030 Resource::OPERATION_SUCCESS);
00030 }
00031
00032 SimpleResource SimpleResourceSystem::alloc() {
00033     SimpleResource rw(RS_NULL_RESOURCE_CODE, &
00034 rs);
00034     rs_resource_code_t code;
00035     code = rs_alloc(&rs);
00036     if (code != RS_NULL_RESOURCE_CODE) {
00037         rw.setCode(code);
00038     }
00039     return rw;
00040 }
00041
00042 SimpleResource SimpleResourceSystem::getResourceByCode
(int code) {
00043     SimpleResource rw((rs_resource_code_t) code, &
00044 rs);
00044     return rw;
00045 }
00046
00047 unsigned int SimpleResourceSystem::totalSpace() {
00048     return (unsigned int) rs_total_space(&rs);
00049 }
00050
00051 unsigned int SimpleResourceSystem::availableSpace() {
00052     return (unsigned int) rs_available_space(&rs);
00053 }
00054
00055 #endif /* __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__ */

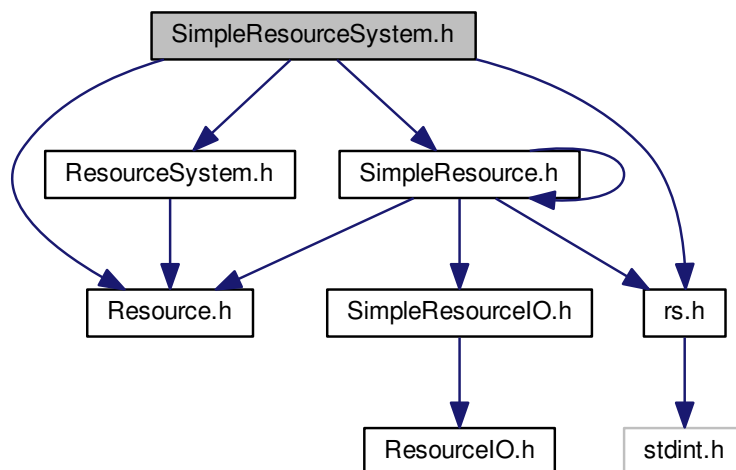
```

5.51 SimpleResourceSystem.h File Reference

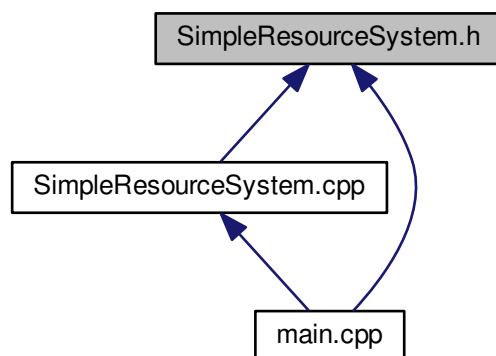
```
#include <Resource.h>
```

```
#include <ResourceSystem.h>
#include <SimpleResource.h>
#include <rs.h>
```

Include dependency graph for SimpleResourceSystem.h:



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



Classes

- class [SimpleResourceSystem](#)

5.52 SimpleResourceSystem.h

00001


```

00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__
00012 #define __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__ 1
00013
00014 #include <Resource.h>
00015 #include <ResourceSystem.h>
00016 #include <SimpleResource.h>
00017 #include <rs.h>
00018
00019 class SimpleResourceSystem : public ResourceSystem {
00020     rs_t rs;
00021     Resource::ResourceOperationResult
        lastOperationResult;
00022 public:
00023
00024     SimpleResourceSystem(int driver);
00025
00026     static bool format(rs_t* rs) {
00027         Resource::ResourceOperationResult o = (
Resource::ResourceOperationResult) rs_format(rs);
00028         return (o == Resource::OPERATION_SUCCESS);
00029     }
00030
00031     rs_t* getRs() {
00032         return &rs;
00033     }
00034
00035     Resource::ResourceOperationResult
getLastOperationResult() {
00036         return lastOperationResult;
00037     }
00038
00039     virtual bool mount(MountOptions options);
00040
00041     virtual bool umount();
00042
00043     SimpleResource alloc();
00044
00045     SimpleResource getResourceByCode(int code);
00046
00047     virtual unsigned int totalSpace();
00048
00049     virtual unsigned int availableSpace();
00050 };
00051
00052 #endif // __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__

```

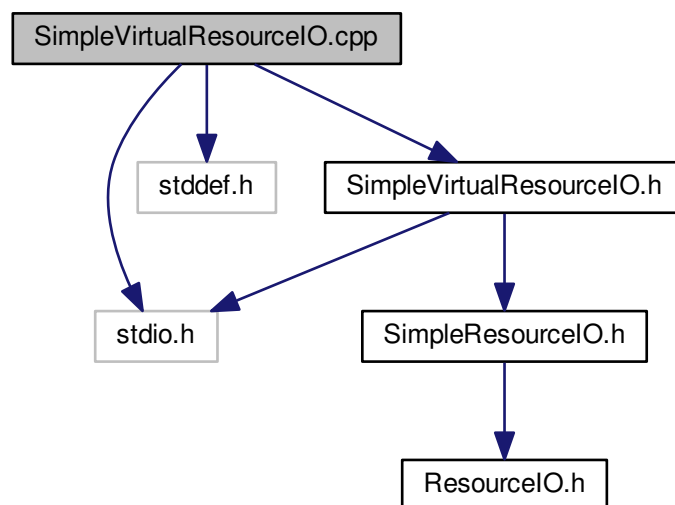
5.53 SimpleVirtualResourceIO.cpp File Reference

```

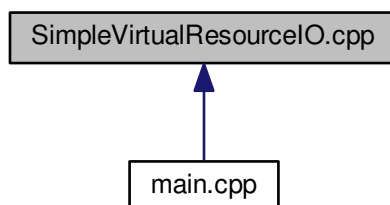
#include <stdio.h>
#include <stddef.h>
#include "SimpleVirtualResourceIO.h"

```

Include dependency graph for SimpleVirtualResourceIO.cpp:



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



Macros

- `#define __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__ 1`

5.53.1 Macro Definition Documentation

5.53.1.1 `#define __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__ 1`

Arduino - A simple resource implementation.

[SimpleVirtualResourceIO.cpp](#)

This is the [Resource](#) IO representation.

Author

Dalmir da Silva dalmirdasilva@gmail.com

Definition at line 12 of file [SimpleVirtualResourceIO.cpp](#).

5.54 SimpleVirtualResourceIO.cpp

```

00001
00011 #ifndef __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__
00012 #define __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__ 1
00013
00014 #include <stdio.h>
00015 #include <stddef.h>
00016 #include "SimpleVirtualResourceIO.h"
00017
00018 SimpleVirtualResourceIO::SimpleVirtualResourceIO(char *
    fileName) : SimpleResourceIO(), fileName(fileName) {
00019     open();
00020 }
00021
00022 bool SimpleVirtualResourceIO::open() {
00023     fp = fopen(fileName, "rb+");
00024     if (fp == NULL) {
00025         printf("Error when opening file: %s.\n", fileName);
00026         exit(1);
00027     }
00028     return true;
00029 }
00030
00031 void SimpleVirtualResourceIO::flush() {
00032     SimpleResourceIO::flush();
00033     fflush(fp);
00034 }
00035
00036 void SimpleVirtualResourceIO::close() {
00037     SimpleResourceIO::close();
00038     fclose(fp);
00039 }
00040
00041 int SimpleVirtualResourceIO::readBytes(unsigned int address, unsigned
    char* buf, int len) {
00042     fseek(fp, address, 0);
00043     return (int) fread(buf, sizeof(unsigned char), len, fp);
00044 }
00045
00046 void SimpleVirtualResourceIO::writeBytes(unsigned int address, unsigned
    char* buf, int len) {
00047     fseek(fp, address, 0);
00048     fwrite(buf, sizeof(unsigned char), len, fp);
00049 }
00050
00051 #endif /* __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__ */
00052

```

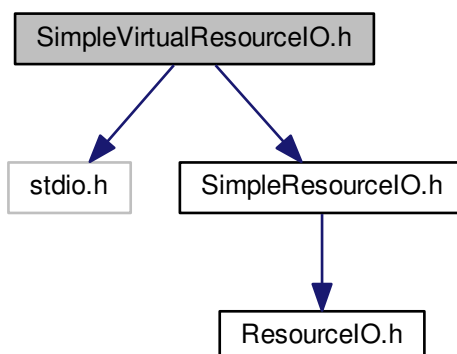
5.55 SimpleVirtualResourceIO.h File Reference

```

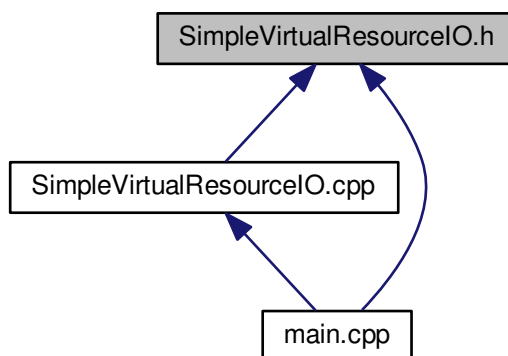
#include <stdio.h>
#include <SimpleResourceIO.h>

```

Include dependency graph for SimpleVirtualResourceIO.h:



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



Classes

- class [SimpleVirtualResourceIO](#)

5.56 SimpleVirtualResourceIO.h

```

00001
00011 #ifndef __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__
00012 #define __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__ 1
00013
00014 #include <stdio.h>
00015 #include <SimpleResourceIO.h>
00016
00017 class SimpleVirtualResourceIO : public SimpleResourceIO {
00018 private:
00019     char *fileName;
  
```

```
00020     FILE *fp;
00021 public:
00022
00023     SimpleVirtualResourceIO(char *fileName);
00024
00025     virtual bool open();
00026
00027     virtual void flush();
00028
00029     virtual void close();
00030 protected:
00031
00032     virtual int readBytes(unsigned int address, unsigned char* buf, int len);
00033
00034     virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
00035 };
00036
00037 #endif /* __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__ */
00038
```


Index

__ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__
 P__
 SimpleArrayResourceIO.cpp, 100

__ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__
 SimpleExternalEepromResourceIO.cpp, 103

__ARDUINO_SIMPLE_RESOURCE_CPP__
 SimpleResource.cpp, 106

__ARDUINO_SIMPLE_RESOURCE_IO_CPP__
 SimpleResourceIO.cpp, 111

__ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__
 SimpleResourceSystem.cpp, 115

__ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__
 SimpleVirtualResourceIO.cpp, 118

__SDCC_RS_C__
 rs.c, 40

__SDCC_RS_INIT_PARTITION_C__
 rs_init_partition.c, 56

__SDCC_RS_IO_C__
 rs_io.c, 61

__SDCC_RS_UTIL_C__
 rs_util.c, 79

_rs_address_to_cluster
 rs_util.h, 89

_rs_address_to_resource_descriptor
 rs_util.h, 90

_rs_alloc_cluster
 rs_util.c, 80
 rs_util.h, 93

_rs_check_for_availability
 rs_util.c, 80
 rs_util.h, 93

_rs_check_for_eor_reached
 rs_util.c, 80
 rs_util.h, 93

_rs_cluster_to_address
 rs_util.h, 90

_rs_create_cluster_chain
 rs_util.c, 80
 rs_util.h, 94

_rs_decrease_free_clusters
 rs_util.h, 90

_rs_format_cluster
 rs_util.c, 80
 rs_util.h, 94

_rs_format_clusters_chain
 rs_util.c, 81
 rs_util.h, 94

_rs_format_resource_descriptor
 rs_util.c, 81
 rs_util.h, 94

_rs_format_resource_clusters
 rs_util.c, 81
 rs_util.h, 94

rs_util.h, 94

_rs_free_cluster
 rs_util.c, 81
 rs_util.h, 95

_rs_free_resource_descriptor
 rs_util.c, 81
 rs_util.h, 95

_rs_free_resource_descriptors
 rs_util.c, 81
 rs_util.h, 95

_rs_has_invalid_attributes
 rs_util.c, 83
 rs_util.h, 95

_rs_increase_free_clusters
 rs_util.h, 90

_rs_io_memory_dump
 main.cpp, 32

_rs_io_read
 rs_io.c, 61
 rs_io.h, 62

_rs_io_write
 rs_io.c, 61
 rs_io.h, 63

_rs_is_driver_monted
 rs_util.c, 83
 rs_util.h, 95

_rs_is_eor_reached
 rs_util.c, 83
 rs_util.h, 95

_rs_is_free_cluster
 rs_util.c, 83
 rs_util.h, 97

_rs_move_current_position_ahead
 rs_util.c, 83
 rs_util.h, 97

_rs_move_current_position_back
 rs_util.c, 84
 rs_util.h, 97

_rs_next_cluster_by_cluster
 rs_util.h, 90

_rs_next_cluster_by_cluster_address
 rs_util.h, 92

_rs_prev_cluster_by_cluster
 rs_util.h, 92

_rs_prev_cluster_by_cluster_address
 rs_util.h, 92

_rs_read_rs_from_disc
 rs_util.c, 84
 rs_util.h, 97

_rs_resource_code_to_resource_descriptor
 rs_util.h, 92

_rs_resource_descriptor_to_address
 rs_util.h, 93

_rs_resource_descriptor_to_resource_code
 rs_util.h, 93

- `_rs_set_driver_monted`
 - `rs_util.c`, [84](#)
 - `rs_util.h`, [98](#)
- `_rs_write_rs_to_disc`
 - `rs_util.c`, [84](#)
 - `rs_util.h`, [98](#)
- `alloc`
 - `SimpleResourceSystem`, [25](#)
- `alloc_resource_spec`
 - `rs_spec.h`, [65](#)
- `allocating_multi_format_spec`
 - `rs_spec.h`, [65](#)
- `array`
 - `SimpleArrayResourceIO`, [14](#)
- `associateIO`
 - `SimpleResourceIO`, [22](#)
- `association`
 - `SimpleResourceIO`, [23](#)
- `availableSpace`
 - `ResourceSystem`, [9](#)
 - `SimpleResourceSystem`, [25](#)
- `CLUSTER_ADDRESS_TO_DATA`
 - `rs.h`, [48](#)
- `CLUSTER_ADDRESS_TO_NEXT`
 - `rs.h`, [48](#)
- `CLUSTER_ADDRESS_TO_PREV`
 - `rs.h`, [48](#)
- `cache`
 - `SimpleResourceIO`, [23](#)
- `cacheHit`
 - `SimpleResourceIO`, [23](#)
- `cacheMemoryAddress`
 - `SimpleResourceIO`, [23](#)
- `cacheMiss`
 - `SimpleResourceIO`, [23](#)
- `checkCache`
 - `SimpleResourceIO`, [22](#)
- `close`
 - `Resource`, [6](#)
 - `ResourceIO`, [7](#)
 - `SimpleResource`, [18](#)
 - `SimpleResourceIO`, [22](#)
 - `SimpleVirtualResourceIO`, [28](#)
- `close_resource_spec`
 - `rs_spec.h`, [65](#)
- `cluster_count`
 - `rs_t`, [11](#)
- `cluster_offset`
 - `rs_resource_t`, [10](#)
- `cluster_table_address`
 - `rs_t`, [11](#)
- `code`
 - `SimpleResource`, [20](#)
- `current_cluster`
 - `rs_resource_t`, [10](#)
- `current_position`
 - `rs_resource_t`, [10](#)
- `driver`
 - `rs_t`, [11](#)
- `driver_mouted`
 - `rs_global_flags_t`, [9](#)
- `eor`
 - `Resource`, [6](#)
 - `SimpleResource`, [18](#)
- `error`
 - `Resource`, [6](#)
 - `SimpleResource`, [18](#)
- `externalEeprom`
 - `SimpleExternalEepromResourceIO`, [16](#)
- `fileName`
 - `SimpleVirtualResourceIO`, [28](#)
- `first_cluster`
 - `rs_resource_t`, [10](#)
- `flags`
 - `rs_resource_t`, [10](#)
 - `rs_stat_t`, [11](#)
 - `rs_t`, [12](#)
- `flush`
 - `ResourceIO`, [7](#)
 - `SimpleResourceIO`, [22](#)
 - `SimpleVirtualResourceIO`, [28](#)
- `format`
 - `SimpleResourceSystem`, [25](#)
- `format_all`
 - `main.c`, [29](#)
- `format_spec`
 - `rs_spec.h`, [65](#)
- `fp`
 - `SimpleVirtualResourceIO`, [28](#)
- `free_clusters`
 - `rs_t`, [12](#)
- `getAssociatedIO`
 - `SimpleResourceIO`, [22](#)
- `getCacheHit`
 - `SimpleResourceIO`, [22](#)
- `getCacheMiss`
 - `SimpleResourceIO`, [22](#)
- `getCode`
 - `SimpleResource`, [18](#)
- `getLastOperationResult`
 - `SimpleResource`, [18](#)
 - `SimpleResourceSystem`, [25](#)
- `getResourceByCode`
 - `SimpleResourceSystem`, [25](#)
- `getRs`
 - `SimpleResourceSystem`, [25](#)
- `isReadOnly`
 - `Resource`, [6](#)
 - `SimpleResource`, [18](#)
- `itob`
 - `main.cpp`, [32](#)
- `lastOperationResult`

- SimpleResource, 20
- SimpleResourceSystem, 26
- MOUNT_READ_ONLY
 - ResourceSystem, 9
- MOUNT_READ_WRITE
 - ResourceSystem, 9
- main
 - main.c, 29
 - main.cpp, 32
- main.c, 28, 29
 - format_all, 29
 - main, 29
 - resource_dump, 29
- main.cpp, 31, 32
 - _rs_io_memory_dump, 32
 - itob, 32
 - main, 32
 - resource_dump, 32
 - VIRTUAL_ENVIROMENT, 32
 - wrapper_format, 32
- memory_size
 - rs_t, 12
- mount
 - ResourceSystem, 9
 - SimpleResourceSystem, 25
- mount_spec
 - rs_spec.h, 65
- MountOptions
 - ResourceSystem, 9
- OPEN_READ_ONLY
 - Resource, 5
- OPEN_READ_WRITE
 - Resource, 5
- OPERATION_ERROR_DRIVER_BUSY
 - Resource, 5
- OPERATION_ERROR_DRIVER_NOT_MOUNTED
 - Resource, 5
- OPERATION_ERROR_NO_SPACE_AVAILABLE
 - Resource, 5
- OPERATION_ERROR_RESOURCE_CLOSED
 - Resource, 5
- OPERATION_ERROR_RESOURCE_DOES_NOT_A↵LLOCATED
 - Resource, 5
- OPERATION_ERROR_RESOURCE_OPENED
 - Resource, 5
- OPERATION_ERROR_RESOURCE_READ_ONLY
 - Resource, 5
- OPERATION_ERROR_SEEK_OUT_OF_BOUND
 - Resource, 5
- OPERATION_SUCCESS
 - Resource, 5
- open
 - Resource, 6
 - ResourceIO, 7
 - SimpleResource, 18
 - SimpleResourceIO, 22
 - SimpleVirtualResourceIO, 28
 - open_resource_spec
 - rs_spec.h, 65
 - OpenOptions
 - Resource, 5
 - print_ln
 - rs_spec_not_virtual.h, 77
 - RD_ADDRESS_TO_FIRST_CLUSTER
 - rs.h, 48
 - RD_ADDRESS_TO_FLAG
 - rs.h, 48
 - RD_ADDRESS_TO_SIZE_HIGH
 - rs.h, 48
 - RD_ADDRESS_TO_SIZE_LOW
 - rs.h, 48
 - RESOURCE_IO_CACHE_SIZE
 - SimpleResourceIO.h, 112
 - RESOURCE_IO_DRIVERS_NUM
 - SimpleResourceIO.h, 112
 - RS_DISK_24K
 - rs_init_partition.h, 58
 - RS_DISK_32K
 - rs_init_partition.h, 58
 - RS_DISK_4K
 - rs_init_partition.h, 58
 - RS_DISK_8K
 - rs_init_partition.h, 58
 - RS_DRIVER_ARDUINO_EEPROM
 - rs.h, 49
 - RS_DRIVER_EXTERNAL_EEPROM
 - rs.h, 49
 - RS_DRIVER_MULTI_EXTERNAL_EEPROM
 - rs.h, 49
 - RS_DRIVER_SELF_EEPROM
 - rs.h, 49
 - RS_DRIVER_VIRTUAL
 - rs.h, 49
 - RS_ENV_ARDUINO
 - rs_init_partition.h, 59
 - RS_ENV_VIRTUAL
 - rs_init_partition.h, 59
 - RS_FIRST_ADDRESS_OF_MEMORY
 - rs.h, 48
 - RS_FLAG_BIT_DRIVER_MOUNTED
 - rs.h, 50
 - RS_FLAG_BIT_READ_ONLY
 - rs.h, 50
 - RS_INEXISTENT_CLUSTER
 - rs.h, 48
 - RS_MOUNT_OPTION_NORMAL
 - rs.h, 50
 - RS_MOUNT_OPTION_READ_ONLY
 - rs.h, 50
 - RS_NULL_CLUSTER
 - rs.h, 48
 - RS_NULL_CLUSTER_ADDRESS
 - rs.h, 48

- RS_NULL_RESOURCE_DESCRIPTOR_ADDRESS
rs.h, [48](#)
- RS_NULL_RESOURCE_CODE
rs.h, [48](#)
- RS_OP_RESULT_ERROR_DRIVER_BUSY
rs.h, [50](#)
- RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED
rs.h, [50](#)
- RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE
rs.h, [50](#)
- RS_OP_RESULT_ERROR_RESOURCE_CLOSED
rs.h, [50](#)
- RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_EXIST
rs.h, [50](#)
- RS_OP_RESULT_ERROR_RESOURCE_OPENED
rs.h, [50](#)
- RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY
rs.h, [50](#)
- RS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUND
rs.h, [50](#)
- RS_OP_RESULT_SUCCESS
rs.h, [50](#)
- RS_OPEN_RESOURCE_OPTION_NORMAL
rs.h, [50](#)
- RS_OPEN_RESOURCE_OPTION_READ_ONLY
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_ALLOCATED
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_EOR_REACHED
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_OPENED
rs.h, [50](#)
- RS_RESOURCE_FLAG_BIT_READ_ONLY
rs.h, [50](#)
- RS_SEEK_ORIGIN_BEGIN
rs.h, [51](#)
- RS_SEEK_ORIGIN_CURRENT
rs.h, [51](#)
- RS_SIZEOF_RESOURCE_SIZE
rs.h, [49](#)
- RS_SPEC_DRIVER
rs_spec.h, [64](#)
- RS_SPEC_IGNORE_0
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_1
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_10
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_11
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_12
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_13
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_14
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_15
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_16
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_17
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_18
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_19
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_2
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_20
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_21
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_22
rs_spec_not_virtual.h, [76](#)
- RS_SPEC_IGNORE_3
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_4
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_5
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_6
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_7
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_8
rs_spec_not_virtual.h, [77](#)
- RS_SPEC_IGNORE_9
rs_spec_not_virtual.h, [77](#)
- random_read_resource_spec
rs_spec.h, [65](#)
- random_read_with_seek_opening_resource_spec
rs_spec.h, [65](#)
- random_read_with_seek_resource_spec
rs_spec.h, [65](#)
- read
Resource, [6](#)
ResourceIO, [8](#)
SimpleResource, [19](#)
SimpleResourceIO, [22](#)
- read_only_mounting_spec
rs_spec.h, [65](#)
- read_only_opening_spec
rs_spec.h, [65](#)
- read_resource_spec
rs_spec.h, [65](#)
- readBytes
Resource, [6](#)

- SimpleArrayResourceIO, 14
- SimpleExternalEepromResourceIO, 16
- SimpleResource, 19
- SimpleResourceIO, 23
- SimpleVirtualResourceIO, 28
- release
 - Resource, 6
 - SimpleResource, 19
- Resource, 4
 - close, 6
 - eor, 6
 - error, 6
 - isReadOnly, 6
 - OPEN_READ_ONLY, 5
 - OPEN_READ_WRITE, 5
 - OPERATION_ERROR_DRIVER_BUSY, 5
 - OPERATION_ERROR_DRIVER_NOT_MOUNTED, 5
 - OPERATION_ERROR_NO_SPACE_AVAILABLE, 5
 - OPERATION_ERROR_RESOURCE_CLOSED, 5
 - OPERATION_ERROR_RESOURCE_DOES_NOT_HAVE_ENOUGH_SPACE, 5
 - OPERATION_ERROR_RESOURCE_OPENED, 5
 - OPERATION_ERROR_RESOURCE_READ_ONLY, 5
 - OPERATION_ERROR_SEEK_OUT_OF_BOUND, 5
 - OPERATION_SUCCESS, 5
 - open, 6
 - OpenOptions, 5
 - read, 6
 - readBytes, 6
 - release, 6
 - ResourceOperationResult, 5
 - ResourceSeekOrigin, 5
 - rewind, 6
 - SEEK_ORIGIN_BEGIN, 5
 - SEEK_ORIGIN_CURRENT, 5
 - seek, 6
 - size, 6
 - sync, 6
 - tell, 6
 - truncate, 6
 - write, 6
 - writeBytes, 6
- resource
 - SimpleResource, 20
- Resource.h, 35, 36
- resource_descriptor
 - rs_resource_t, 10
- resource_descriptor_count
 - rs_t, 12
- resource_descriptor_table_address
 - rs_t, 12
- resource_dump
 - main.c, 29
 - main.cpp, 32
- ResourceIO, 7
 - close, 7
 - flush, 7
 - open, 7
 - read, 8
 - write, 8
- ResourceIO.h, 37
- ResourceOperationResult
 - Resource, 5
- ResourceSeekOrigin
 - Resource, 5
- ResourceSystem, 8
 - availableSpace, 9
 - MOUNT_READ_ONLY, 9
 - MOUNT_READ_WRITE, 9
 - mount, 9
 - MountOptions, 9
 - totalSpace, 9
 - umount, 9
- ResourceSystem.h, 37, 38
- rewind
 - Resource, 6
 - SimpleResource, 19
- rewind_resource_spec
 - rs_spec.h, 65
- rs
 - SimpleResource, 20
 - SimpleResourceSystem, 26
- rs.c, 38, 42
 - __SDCC_RS_C__, 40
 - rs_alloc, 40
 - rs_available_space, 40
 - rs_close, 40
 - rs_eor, 40
 - rs_error, 40
 - rs_format, 41
 - rs_global_flags, 42
 - rs_mount, 41
 - rs_open, 41
 - rs_read, 41
 - rs_release, 41
 - rs_rewind, 41
 - rs_seek, 41
 - rs_size, 41
 - rs_stat, 41
 - rs_sync, 41
 - rs_tell, 41
 - rs_total_space, 41
 - rs_truncate, 41
 - rs_umount, 41
 - rs_write, 41
- rs.h, 46, 52
 - CLUSTER_ADDRESS_TO_DATA, 48
 - CLUSTER_ADDRESS_TO_NEXT, 48
 - CLUSTER_ADDRESS_TO_PREV, 48
 - RD_ADDRESS_TO_FIRST_CLUSTER, 48
 - RD_ADDRESS_TO_FLAG, 48
 - RD_ADDRESS_TO_SIZE_HIGH, 48

RD_ADDRESS_TO_SIZE_LOW, 48
 RS_DRIVER_ARDUINO_EEPROM, 49
 RS_DRIVER_EXTERNAL_EEPROM, 49
 RS_DRIVER_MULTI_EXTERNAL_EEPROM, 49
 RS_DRIVER_SELF_EEPROM, 49
 RS_DRIVER_VIRTUAL, 49
 RS_FIRST_ADDRESS_OF_MEMORY, 48
 RS_FLAG_BIT_DRIVER_MOUNTED, 50
 RS_FLAG_BIT_READ_ONLY, 50
 RS_INEXISTENT_CLUSTER, 48
 RS_MOUNT_OPTION_NORMAL, 50
 RS_MOUNT_OPTION_READ_ONLY, 50
 RS_NULL_CLUSTER, 48
 RS_NULL_CLUSTER_ADDRESS, 48
 RS_NULL_RESOURCE_DESCRIPTOR_ADDRESS, 48
 RS_NULL_RESOURCE_CODE, 48
 RS_OP_RESULT_ERROR_DRIVER_BUSY, 50
 RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED, 50
 RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE, 50
 RS_OP_RESULT_ERROR_RESOURCE_CLOSED, 50
 RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_EXIST, 50
 RS_OP_RESULT_ERROR_RESOURCE_OPENED, 50
 RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY, 50
 RS_OP_RESULT_ERROR_SEEK_OUT_OF_BOUNDS, 50
 RS_OP_RESULT_SUCCESS, 50
 RS_OPEN_RESOURCE_OPTION_NORMAL, 50
 RS_OPEN_RESOURCE_OPTION_READ_ONLY, 50
 RS_RESOURCE_FLAG_BIT_ALLOCATED, 50
 RS_RESOURCE_FLAG_BIT_EOR_REACHED, 50
 RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ, 50
 RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE, 50
 RS_RESOURCE_FLAG_BIT_OPENED, 50
 RS_RESOURCE_FLAG_BIT_READ_ONLY, 50
 RS_SEEK_ORIGIN_BEGIN, 51
 RS_SEEK_ORIGIN_CURRENT, 51
 RS_SIZEOF_RESOURCE_SIZE, 49
 rs_alloc, 51
 rs_available_space, 51
 rs_close, 51
 rs_cluster_t, 49
 rs_driver_t, 49
 rs_eor, 51
 rs_error, 51
 rs_flag_bits_t, 49
 rs_format, 51
 rs_global_flags, 52
 rs_memory_address_t, 49
 rs_mount, 51
 rs_mount_options_t, 50
 rs_op_result_t, 50
 rs_open, 51
 rs_open_resource_options_t, 50
 rs_read, 51
 rs_release, 51
 rs_resource_code_t, 49
 rs_resource_descriptor_t, 49
 rs_resource_flag_bits_t, 50
 rs_resource_size_t, 49
 rs_rewind, 51
 rs_seek, 51
 rs_seek_int_t, 49
 rs_seek_origin_t, 50
 rs_size, 51
 rs_stat, 52
 rs_sync, 52
 rs_tell, 52
 rs_total_space, 52
 rs_truncate, 52
 rs_umount, 52
 rs_write, 52
 rs_alloc
 rs.c, 40
 rs.h, 51
 rs_available_space
 rs.c, 40
 rs.h, 51
 rs_close
 rs.c, 40
 rs.h, 51
 rs_cluster_t
 rs.h, 49
 rs_disk_size_t
 rs_init_partition.h, 58
 rs_driver_t
 rs.h, 49
 rs_environment_t
 rs_init_partition.h, 58
 rs_eor
 rs.c, 40
 rs.h, 51
 rs_error
 rs.c, 40
 rs.h, 51
 rs_flag_bits_t
 rs.h, 49
 rs_format
 rs.c, 41
 rs.h, 51
 rs_global_flags
 rs.c, 42
 rs.h, 52
 rs_global_flags_t, 9
 driver_mounted, 9
 rs_init_partition

- rs_init_partition.c, 56
- rs_init_partition.h, 59
- rs_init_partition.c, 55, 56
 - __SDCC_RS_INIT_PARTITION_C__, 56
 - rs_init_partition, 56
- rs_init_partition.h, 57, 59
 - RS_DISK_24K, 58
 - RS_DISK_32K, 58
 - RS_DISK_4K, 58
 - RS_DISK_8K, 58
 - RS_ENV_ARDUINO, 59
 - RS_ENV_VIRTUAL, 59
 - rs_disk_size_t, 58
 - rs_environment_t, 58
 - rs_init_partition, 59
- rs_io.c, 60, 61
 - __SDCC_RS_IO_C__, 61
 - _rs_io_read, 61
 - _rs_io_write, 61
- rs_io.h, 61, 63
 - _rs_io_read, 62
 - _rs_io_write, 63
- rs_memory_address_t
 - rs.h, 49
- rs_mount
 - rs.c, 41
 - rs.h, 51
- rs_mount_options_t
 - rs.h, 50
- rs_op_result_t
 - rs.h, 50
- rs_open
 - rs.c, 41
 - rs.h, 51
- rs_open_resource_options_t
 - rs.h, 50
- rs_read
 - rs.c, 41
 - rs.h, 51
- rs_release
 - rs.c, 41
 - rs.h, 51
- rs_resource_code_t
 - rs.h, 49
- rs_resource_descriptor_t
 - rs.h, 49
- rs_resource_flag_bits_t
 - rs.h, 50
- rs_resource_size_t
 - rs.h, 49
- rs_resource_t, 10
 - cluster_offset, 10
 - current_cluster, 10
 - current_position, 10
 - first_cluster, 10
 - flags, 10
 - resource_descriptor, 10
 - size, 10
- rs_rewind
 - rs.c, 41
 - rs.h, 51
- rs_seek
 - rs.c, 41
 - rs.h, 51
- rs_seek_int_t
 - rs.h, 49
- rs_seek_origin_t
 - rs.h, 50
- rs_size
 - rs.c, 41
 - rs.h, 51
- rs_spec.h, 63, 66
 - alloc_resource_spec, 65
 - allocating_multi_format_spec, 65
 - close_resource_spec, 65
 - format_spec, 65
 - mount_spec, 65
 - open_resource_spec, 65
 - RS_SPEC_DRIVER, 64
 - random_read_resource_spec, 65
 - random_read_with_seek_opening_resource_spec, 65
 - random_read_with_seek_resource_spec, 65
 - read_only_mounting_spec, 65
 - read_only_opening_spec, 65
 - read_resource_spec, 65
 - rewind_resource_spec, 65
 - rs_spec_printf, 64
 - seek_resource_spec, 65
 - size_resource_spec, 65
 - tell_resource_spec, 66
 - tell_with_seek_resource_spec, 66
 - total_space_resource_spec, 66
 - try_read_when_end_of_resource_is_reached_spec, 66
 - try_read_when_resource_is_closed_spec, 66
 - try_to_alloc_resources_that_is_possible_spec, 66
 - umount_spec, 66
 - write_resource_spec, 66
- rs_spec_not_virtual.h, 74, 77
 - print_ln, 77
 - RS_SPEC_IGNORE_0, 76
 - RS_SPEC_IGNORE_1, 76
 - RS_SPEC_IGNORE_10, 76
 - RS_SPEC_IGNORE_11, 76
 - RS_SPEC_IGNORE_12, 76
 - RS_SPEC_IGNORE_13, 76
 - RS_SPEC_IGNORE_14, 76
 - RS_SPEC_IGNORE_15, 76
 - RS_SPEC_IGNORE_16, 76
 - RS_SPEC_IGNORE_17, 76
 - RS_SPEC_IGNORE_18, 76
 - RS_SPEC_IGNORE_19, 76
 - RS_SPEC_IGNORE_2, 76
 - RS_SPEC_IGNORE_20, 76
 - RS_SPEC_IGNORE_21, 76

- RS_SPEC_IGNORE_22, 76
- RS_SPEC_IGNORE_3, 77
- RS_SPEC_IGNORE_4, 77
- RS_SPEC_IGNORE_5, 77
- RS_SPEC_IGNORE_6, 77
- RS_SPEC_IGNORE_7, 77
- RS_SPEC_IGNORE_8, 77
- RS_SPEC_IGNORE_9, 77
- rs_spec_printf, 77
- rs_spec_printf
 - rs_spec.h, 64
 - rs_spec_not_virtual.h, 77
- rs_stat
 - rs.c, 41
 - rs.h, 52
- rs_stat_t, 10
 - flags, 11
- rs_sync
 - rs.c, 41
 - rs.h, 52
- rs_t, 11
 - cluster_count, 11
 - cluster_table_address, 11
 - driver, 11
 - flags, 12
 - free_clusters, 12
 - memory_size, 12
 - resource_descriptor_count, 12
 - resource_descriptor_table_address, 12
 - sizeof_cluster, 12
 - sizeof_cluster_control, 12
 - sizeof_cluster_data, 12
 - sizeof_cluster_table, 12
 - sizeof_resource_descriptor, 12
 - sizeof_resource_descriptor_table, 12
- rs_tell
 - rs.c, 41
 - rs.h, 52
- rs_total_space
 - rs.c, 41
 - rs.h, 52
- rs_truncate
 - rs.c, 41
 - rs.h, 52
- rs_umount
 - rs.c, 41
 - rs.h, 52
- rs_util.c, 78, 85
 - __SDCC_RS_UTIL_C__, 79
 - _rs_alloc_cluster, 80
 - _rs_check_for_availability, 80
 - _rs_check_for_eor_reached, 80
 - _rs_create_cluster_chain, 80
 - _rs_format_cluster, 80
 - _rs_format_clusters_chain, 81
 - _rs_format_resource_descriptor, 81
 - _rs_format_resource_clusters, 81
 - _rs_free_cluster, 81
 - _rs_free_resource_descriptor, 81
 - _rs_free_resource_descriptors, 81
 - _rs_has_invalid_attributes, 83
 - _rs_is_driver_monted, 83
 - _rs_is_eor_reached, 83
 - _rs_is_free_cluster, 83
 - _rs_move_current_position_ahead, 83
 - _rs_move_current_position_back, 84
 - _rs_read_rs_from_disc, 84
 - _rs_set_driver_monted, 84
 - _rs_write_rs_to_disc, 84
- rs_util.h, 88, 98
 - _rs_address_to_cluster, 89
 - _rs_address_to_resource_descriptor, 90
 - _rs_alloc_cluster, 93
 - _rs_check_for_availability, 93
 - _rs_check_for_eor_reached, 93
 - _rs_cluster_to_address, 90
 - _rs_create_cluster_chain, 94
 - _rs_decrease_free_clusters, 90
 - _rs_format_cluster, 94
 - _rs_format_clusters_chain, 94
 - _rs_format_resource_descriptor, 94
 - _rs_format_resource_clusters, 94
 - _rs_free_cluster, 95
 - _rs_free_resource_descriptor, 95
 - _rs_free_resource_descriptors, 95
 - _rs_has_invalid_attributes, 95
 - _rs_increase_free_clusters, 90
 - _rs_is_driver_monted, 95
 - _rs_is_eor_reached, 95
 - _rs_is_free_cluster, 97
 - _rs_move_current_position_ahead, 97
 - _rs_move_current_position_back, 97
 - _rs_next_cluster_by_cluster, 90
 - _rs_next_cluster_by_cluster_address, 92
 - _rs_prev_cluster_by_cluster, 92
 - _rs_prev_cluster_by_cluster_address, 92
 - _rs_read_rs_from_disc, 97
 - _rs_resource_code_to_resource_descriptor, 92
 - _rs_resource_descriptor_to_address, 93
 - _rs_resource_descriptor_to_resource_code, 93
 - _rs_set_driver_monted, 98
 - _rs_write_rs_to_disc, 98
- rs_util_spec.h, 99
- rs_write
 - rs.c, 41
 - rs.h, 52
- SEEK_ORIGIN_BEGIN
 - Resource, 5
- SEEK_ORIGIN_CURRENT
 - Resource, 5
- seek
 - Resource, 6
 - SimpleResource, 19
- seek_resource_spec
 - rs_spec.h, 65
- setCode

- SimpleResource, 19
- SimpleArrayResourceIO, 12
 - array, 14
 - readBytes, 14
 - SimpleArrayResourceIO, 14
 - size, 14
 - writeBytes, 14
- SimpleArrayResourceIO.cpp, 99, 101
 - __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_CPP__, 100
- SimpleArrayResourceIO.h, 101, 102
- SimpleExternalEepromResourceIO, 14
 - externalEeprom, 16
 - readBytes, 16
 - SimpleExternalEepromResourceIO, 16
 - writeBytes, 16
- SimpleExternalEepromResourceIO.cpp, 102, 104
 - __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__, 103
- SimpleExternalEepromResourceIO.h, 104, 105
- SimpleResource, 16
 - close, 18
 - code, 20
 - eor, 18
 - error, 18
 - getCode, 18
 - getLastOperationResult, 18
 - isReadOnly, 18
 - lastOperationResult, 20
 - open, 18
 - read, 19
 - readBytes, 19
 - release, 19
 - resource, 20
 - rewind, 19
 - rs, 20
 - seek, 19
 - setCode, 19
 - SimpleResource, 18
 - size, 19
 - sync, 19
 - tell, 19
 - truncate, 19
 - write, 19
 - writeBytes, 20
- SimpleResource.cpp, 105, 107
 - __ARDUINO_SIMPLE_RESOURCE_CPP__, 106
- SimpleResource.h, 108, 109
- SimpleResourceIO, 20
 - associateIO, 22
 - association, 23
 - cache, 23
 - cacheHit, 23
 - cacheMemoryAddress, 23
 - cacheMiss, 23
 - checkCache, 22
 - close, 22
 - flush, 22
 - getAssociatedIO, 22
 - getCacheHit, 22
 - getCacheMiss, 22
 - open, 22
 - read, 22
 - readBytes, 23
 - SimpleResourceIO, 22
 - validCacheSize, 23
 - wasCacheChanged, 23
 - wasCacheInitialized, 23
 - write, 23
 - writeBytes, 23
- SimpleResourceIO.cpp, 110, 111
 - __ARDUINO_SIMPLE_RESOURCE_IO_CPP__, 111
- SimpleResourceIO.h, 112, 113
 - RESOURCE_IO_CACHE_SIZE, 112
 - RESOURCE_IO_DRIVERS_NUM, 112
- SimpleResourceSystem, 24
 - alloc, 25
 - availableSpace, 25
 - format, 25
 - getLastOperationResult, 25
 - getResourceByCode, 25
 - getRs, 25
 - lastOperationResult, 26
 - mount, 25
 - rs, 26
 - SimpleResourceSystem, 25
 - totalSpace, 25
 - umount, 26
- SimpleResourceSystem.cpp, 114, 115
 - __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__, 115
- SimpleResourceSystem.h, 115, 116
- SimpleVirtualResourceIO, 26
 - close, 28
 - fileName, 28
 - flush, 28
 - fp, 28
 - open, 28
 - readBytes, 28
 - SimpleVirtualResourceIO, 28
 - writeBytes, 28
- SimpleVirtualResourceIO.cpp, 117, 119
 - __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_CPP__, 118
- SimpleVirtualResourceIO.h, 119, 120
- size
 - Resource, 6
 - rs_resource_t, 10
 - SimpleArrayResourceIO, 14
 - SimpleResource, 19
 - size_resource_spec
 - rs_spec.h, 65
 - sizeof_cluster
 - rs_t, 12
 - sizeof_cluster_control

- rs_t, [12](#)
- sizeof_cluster_data
 - rs_t, [12](#)
- sizeof_cluster_table
 - rs_t, [12](#)
- sizeof_resource_descriptor
 - rs_t, [12](#)
- sizeof_resource_descriptor_table
 - rs_t, [12](#)
- sync
 - Resource, [6](#)
 - SimpleResource, [19](#)
- tell
 - Resource, [6](#)
 - SimpleResource, [19](#)
- tell_resource_spec
 - rs_spec.h, [66](#)
- tell_with_seek_resource_spec
 - rs_spec.h, [66](#)
- total_space_resource_spec
 - rs_spec.h, [66](#)
- totalSpace
 - ResourceSystem, [9](#)
 - SimpleResourceSystem, [25](#)
- truncate
 - Resource, [6](#)
 - SimpleResource, [19](#)
- try_read_when_end_of_resource_is_reached_spec
 - rs_spec.h, [66](#)
- try_read_when_resource_is_closed_spec
 - rs_spec.h, [66](#)
- try_to_alloc_resources_that_is_possible_spec
 - rs_spec.h, [66](#)
- umount
 - ResourceSystem, [9](#)
 - SimpleResourceSystem, [26](#)
- umount_spec
 - rs_spec.h, [66](#)
- VIRTUAL_ENVIROMENT
 - main.cpp, [32](#)
- validCacheSize
 - SimpleResourceIO, [23](#)
- wasCacheChanged
 - SimpleResourceIO, [23](#)
- wasCacheInitialized
 - SimpleResourceIO, [23](#)
- wrapper_format
 - main.cpp, [32](#)
- write
 - Resource, [6](#)
 - ResourceIO, [8](#)
 - SimpleResource, [19](#)
 - SimpleResourceIO, [23](#)
- write_resource_spec
 - rs_spec.h, [66](#)
- writeBytes
 - Resource, [6](#)
 - SimpleArrayResourceIO, [14](#)
 - SimpleExternalEepromResourceIO, [16](#)
 - SimpleResource, [20](#)
 - SimpleResourceIO, [23](#)
 - SimpleVirtualResourceIO, [28](#)