Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:52:22

Contents

1 Hierarchical Index

1.1 Class Hierarchy

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

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

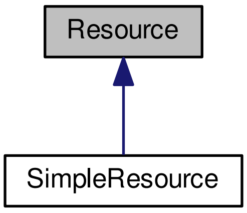
Here is a list of all files with brief descriptions:

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, OPE↩
  RATION_ERROR_SEEK_OUT_OF_BOUND = 6, OPERATION_ERROR_RESOURCE_DOES_NOT_ALL↩
  OCATED = 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 Resource::writeBytes (unsigned char * *buf*, int *len*) `[pure virtual]`

Implemented in SimpleResource.

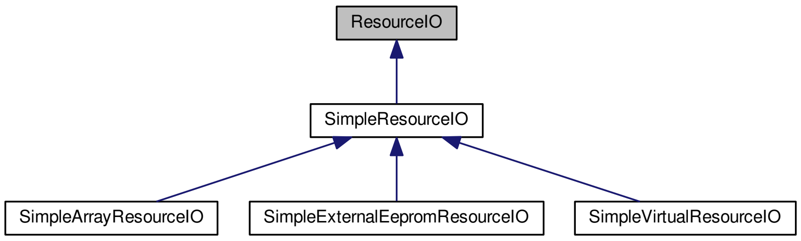
The documentation for this class was generated from the following file:

- 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.

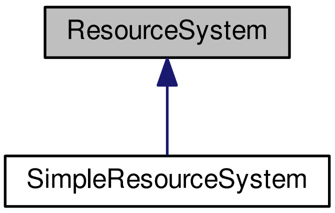
The documentation for this class was generated from the following file:

- 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.

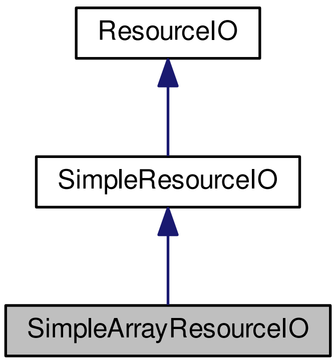
The documentation for this struct was generated from the following file:

- rs.h

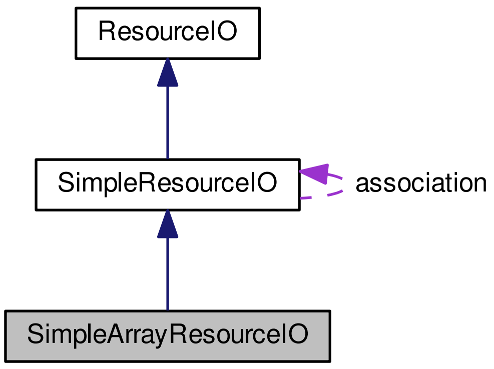
4.8 SimpleArrayResourceIO Class Reference

```
#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.

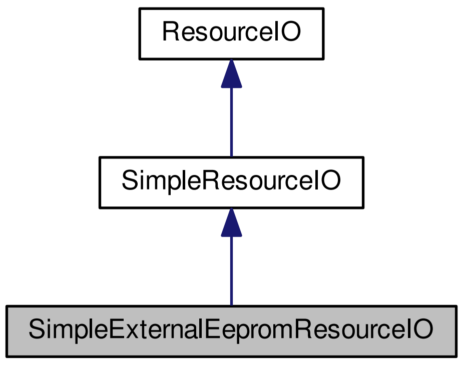
The documentation for this class was generated from the following files:

- SimpleArrayResourceIO.h
- SimpleArrayResourceIO.cpp

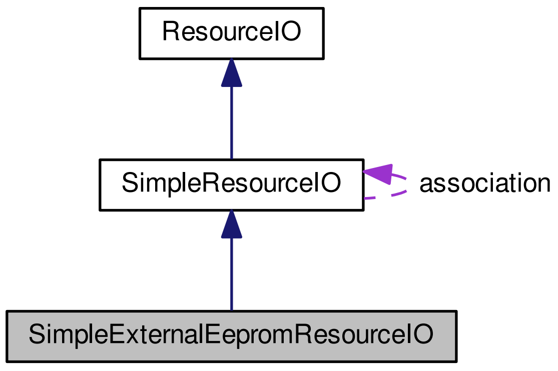
4.9 SimpleExternalEepromResourceIO Class Reference

```
#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.

SimpleExternlaEepromResourceIO.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.

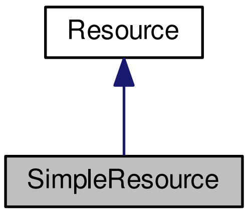
The documentation for this class was generated from the following files:

- SimpleExternalEepromResourceIO.h
- SimpleExternalEepromResourceIO.cpp

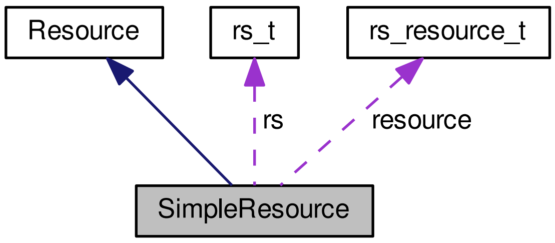
4.10 SimpleResource Class Reference

```
#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.

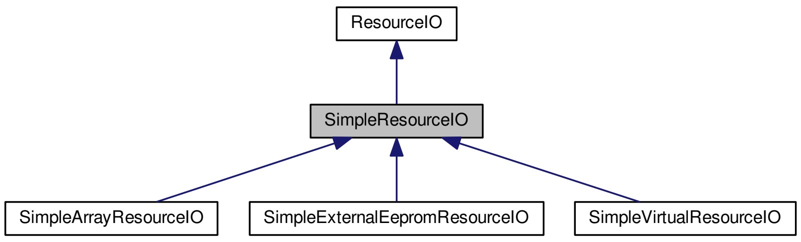
The documentation for this class was generated from the following files:

- SimpleResource.h
- SimpleResource.cpp

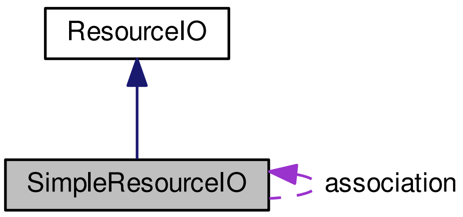
4.11 SimpleResourceIO Class Reference

```
#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:

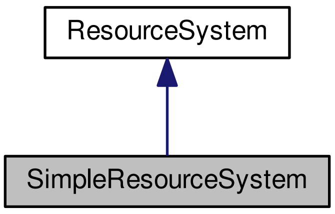
- SimpleResourceIO.h

- SimpleResourceIO.cpp

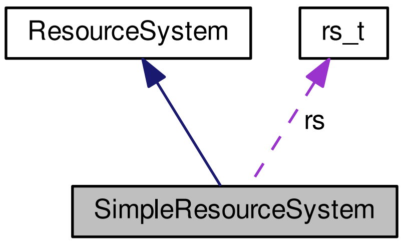
4.12 SimpleResourceSystem Class Reference

`#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.

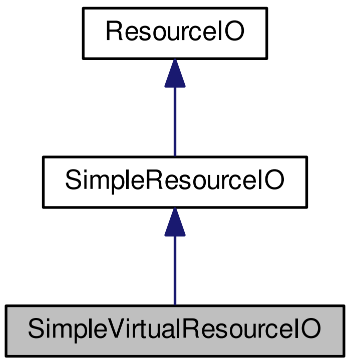
The documentation for this class was generated from the following files:

- SimpleResourceSystem.h
- SimpleResourceSystem.cpp

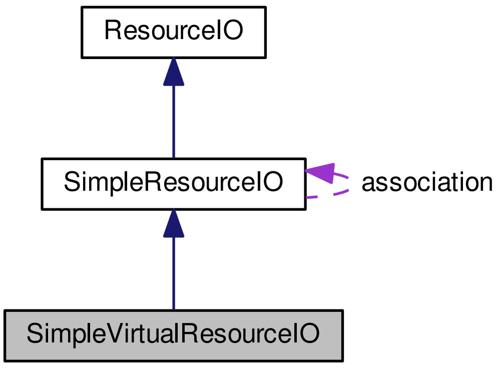
4.13 SimpleVirtualResourceIO Class Reference

`#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

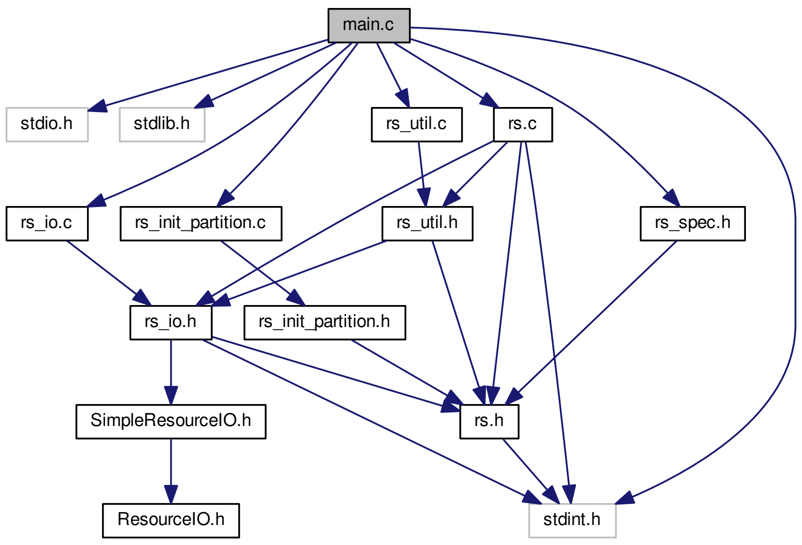
5 File Documentation

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

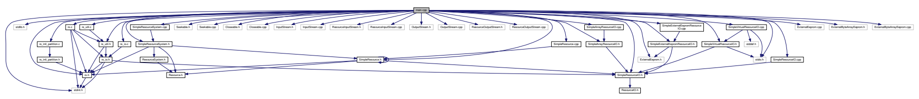
```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <stdint.h>
00004
```

```c
#include "rs_io.c"
#include "rs_util.c"
#include "rs.c"
#include "rs_init_partition.c"

void resource_dump(rs_resource_t *resource);

#include "rs_spec.h"

void format_all() {
    uint16_t i;
    for (i = 0; i < 0x7fff; i++) {
        _rs_io_write(RS_DRIVER_VIRTUAL, i, 0x00);
    }
}

int main() {
    rs_t rs;

    format_all();
    rs_init_partition(&rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    rs_format(&rs);
    format_spec(&rs);
    mount_spec(&rs);
    umount_spec(&rs);
    alloc_resource_spec(&rs);
    try_to_alloc_resources_that_is_possible_spec(&rs);
    open_resource_spec(&rs);
    write_resource_spec(&rs);
    rewind_resource_spec(&rs);
    read_resource_spec(&rs);
    close_resource_spec(&rs);
    try_read_when_end_of_resource_is_reached_spec(&rs);
    try_read_when_resource_is_closed_spec(&rs);
    seek_resource_spec(&rs);
    random_read_resource_spec(&rs);
    random_read_with_seek_resource_spec(&rs);
    random_read_with_seek_opening_resource_spec(&rs);
    size_resource_spec(&rs);
    tell_resource_spec(&rs);
    tell_with_seek_resource_spec(&rs);
    total_space_resource_spec(&rs);
    allocating_multi_format_spec(&rs);
    read_only_mounting_spec(&rs);
    read_only_opening_spec(&rs);

    rs_mount(RS_SPEC_DRIVER, &rs, RS_MOUNT_OPTION_NORMAL);
    _rs_io_memory_dump(&rs);
    return 0;
}

void resource_dump(rs_resource_t *resource) {
    printf("======== resource dump begin ========\n");
    printf("  resource descriptor: %6d %s\n", resource->resource_descriptor, itob(resource->resource_descriptor));
    printf("  first cluster:______ %6d %s\n", resource->first_cluster, itob(resource->first_cluster));
    printf("  current cluster:____ %6d %s\n", resource->current_cluster, itob(resource->current_cluster));
    printf("  cluster offset:_____ %6d %s\n", resource->cluster_offset, itob(resource->cluster_offset));
    printf("  size:_______________ %6d %s\n", resource->size, itob(resource->size));
    printf("  current position:___ %6d %s\n", resource->current_position, itob(resource->current_position));
    printf("  flags:______________ %6d %s\n", resource->flags, itob(resource->flags));
    printf("  errors:_____________ %6d %s\n", rs_error(resource), itob(rs_error(resource)));
    printf("========= resource dump end =========\n");
}
```

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.

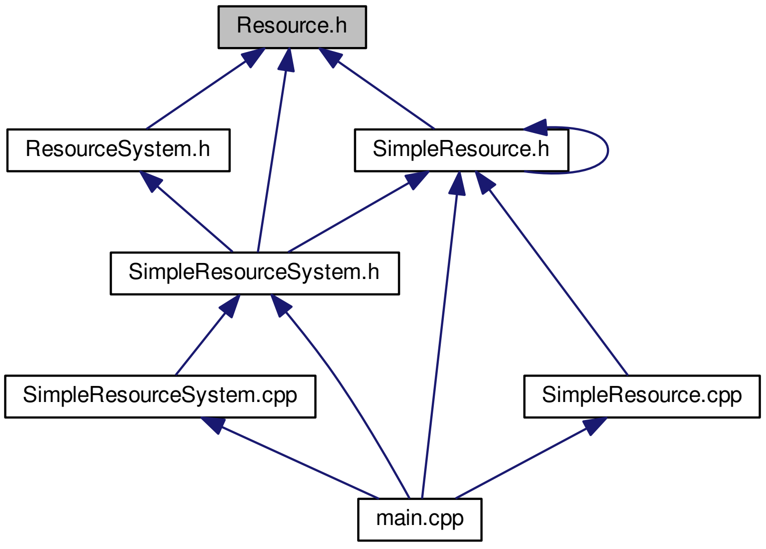
5.4 main.cpp

```
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>
```

```
#include <ExternalByteArrayEeprom.cpp>

char* itob(int i) {
    int bits;
    int j, k;
    uint16_t mi = 0;
    mi |= i;
    static char buff[sizeof (mi) * 8 + 1];
    bits = sizeof (mi) * 8;
    for (j = bits - 1, k = 0; j >= 0; j--, k++) {
        buff[k] = ((mi >> j) & 0x01) + '0';
    }
    buff[bits] = '\0';
    return buff;
}

void _rs_io_memory_dump(rs_t *rs) {
    rs_memory_address_t memory_address;
    uint16_t count, count2;
    uint8_t d = 0;
    FILE *fp;
    if (!_rs_is_driver_monted(rs->driver)) {
        printf("Rs not mouted yet\n");
        return;
    }
    fp = fopen("dump", "w+");
    fprintf(fp, "DRIVER: %x\n", rs->driver);
    fprintf(fp, "\n==========================\n");
    fprintf(fp, "\nRs\n");
    fprintf(fp, "------------------\n");
    fprintf(fp, "memory_size:                        0x%04x %4d %s\n", rs->
memory_size, rs->memory_size, itob(rs->memory_size));
    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));
    fprintf(fp, "cluster_table_address:              0x%04x %4d %s\n", rs->
cluster_table_address, rs->cluster_table_address,
itob(rs->cluster_table_address));
    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));
    fprintf(fp, "sizeof_cluster_table:               0x%04x %4d %s\n", rs->
sizeof_cluster_table, rs->sizeof_cluster_table,
itob(rs->sizeof_resource_descriptor_table));
    fprintf(fp, "sizeof_resource_descriptor:         0x%04x %4d %s\n", rs->
sizeof_resource_descriptor, rs->
sizeof_resource_descriptor, itob(rs->
sizeof_resource_descriptor));
    fprintf(fp, "sizeof_cluster:                     0x%04x %4d %s\n", rs->
sizeof_cluster, rs->sizeof_cluster, itob(rs->
sizeof_cluster));
    fprintf(fp, "resource_descriptor_count:          0x%04x %4d %s\n", rs->
resource_descriptor_count, rs->
resource_descriptor_count, itob(rs->
resource_descriptor_count));
    fprintf(fp, "cluster_count:                      0x%04x %4d %s\n", rs->
cluster_count, rs->cluster_count, itob(rs->
cluster_count));
    fprintf(fp, "sizeof_cluster_data:                0x%04x %4d %s\n", rs->
sizeof_cluster_data, rs->sizeof_cluster_data,
itob(rs->sizeof_cluster_data));
    fprintf(fp, "sizeof_cluster_control:             0x%04x %4d %s\n", rs->
sizeof_cluster_control, rs->sizeof_cluster_control,
itob(rs->sizeof_cluster_control));
    fprintf(fp, "free_clusters:                      0x%04x %4d %s\n", rs->
free_clusters, rs->free_clusters, itob(rs->
free_clusters));
    fprintf(fp, "flags:                              0x%04x %4d %s\n", rs->flags, rs->
flags, itob(rs->flags));
    fprintf(fp, "\n==========================\n");
    fprintf(fp, "\nResource table\n");
    fprintf(fp, "------------------\n");
    count = 0;
    for (
        memory_address = rs->resource_descriptor_table_address;
        memory_address < (rs->resource_descriptor_table_address + rs->
sizeof_resource_descriptor_table);
        memory_address++
        ) {

        if ((count % rs->sizeof_resource_descriptor) == 0) {
            fprintf(fp, "\n%02x: ", (count) ? count / rs->
sizeof_resource_descriptor : 0);
        }
        fprintf(fp, "%02x ", _rs_io_read(rs->driver, memory_address));
        count++;
    }
    fprintf(fp, "\n==========================\n");
    fprintf(fp, "\nCluster table\n");
    fprintf(fp, "-----------------\n");
    fprintf(fp, "\n    |nn |pp |");
    for (count = 0; count < rs->sizeof_cluster_data; count++) {
        fprintf(fp, "dd ");
    }
    fprintf(fp, "\n    ---------");
    for (count = 0; count < rs->sizeof_cluster_data; count++) {
        fprintf(fp, "---");
    }
    count = 0;

    for (
        memory_address = rs->cluster_table_address;
        memory_address < (rs->cluster_table_address + rs->sizeof_cluster_table);
        memory_address++
        ) {
        if ((count % rs->sizeof_cluster) == 0) {
            fprintf(fp, "\n%02x: |", (count) ? count / rs->sizeof_cluster : 0);
            count2 = 0;
        }
        if (count2 == 1 || count2 == 2) {
            fprintf(fp, "|");
        }
        fprintf(fp, "%02x ", (d = _rs_io_read(rs->driver, memory_address)));
        fflush(fp);
        count++;
        count2++;
    }
    fclose(fp);
}

void resource_dump(rs_resource_t *resource) {
    printf("======== resource dump begin ========\n");
    printf("  resource descriptor: %6d %s\n", resource->resource_descriptor, itob(resource->resource_descriptor));
    printf("  first cluster:______ %6d %s\n", resource->first_cluster, itob(resource->first_cluster));
    printf("  current cluster:____ %6d %s\n", resource->current_cluster, itob(resource->current_cluster));
    printf("  cluster offset:_____ %6d %s\n", resource->cluster_offset, itob(resource->cluster_offset));
    printf("  size:_______________ %6d %s\n", resource->size, itob(resource->size));
    printf("  current position:___ %6d %s\n", resource->current_position, itob(resource->current_position));
    printf("  flags:______________ %6d %s\n", resource->flags, itob(resource->flags));
    printf("  errors:_____________ %6d %s\n", rs_error(resource), itob(rs_error(resource)));
    printf("========= resource dump end =========\n");
}

void wrapper_format(rs_t *rs) {
    uint8_t b[5] = {0xf0, 0x01, 0xff, 0xdd, 0xfa};
    uint8_t a[32768];
    uint8_t c[32768];
    rs->driver = RS_DRIVER_VIRTUAL;
    printf("format: %x\n", SimpleResourceSystem::format(rs));
    SimpleResourceSystem rsw(RS_DRIVER_VIRTUAL);
    printf("mount: %x\n", rsw.mount(ResourceSystem::MOUNT_READ_WRITE));
    SimpleResource rw = rsw.alloc();
    printf("code: %x\n", rw.getCode());
    printf("open: %d\n", rw.open(Resource::OPEN_READ_WRITE));
    rw.writeBytes(b, 5);
    rw.rewind();
    printf("read: %x\n", rw.read());
    printf("size: %x\n", rw.size());
    rw.seek(Resource::SEEK_ORIGIN_BEGIN, 2);
    printf("read: %x\n", rw.read());
    printf("availableSpace: %x\n", rsw.availableSpace());
    rw.release();
    printf("availableSpace: %x\n", rsw.availableSpace());
    printf("close: %x\n", rw.close());
    printf("open: %d\n", rw.open(Resource::OPEN_READ_WRITE));
    printf("lor: %d\n", rw.getLastOperationResult());
}

int main() {
    rs_t rs;
    char *s = (char *) "/tmp/img.bin";
    SimpleVirtualResourceIO io(s);
    printf("Running wrapper specs...\n");

    rs_init_partition(&rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    SimpleResourceIO::associateIO(&io,
RS_DRIVER_VIRTUAL);

    //wrapper_format(&rs);

    printf("Format: %d\n", SimpleResourceSystem::format(&rs));
    SimpleResourceSystem srs(RS_DRIVER_VIRTUAL);
    printf("Mount: %d\n", srs.mount(ResourceSystem::MOUNT_READ_WRITE))
;
    SimpleResource rw = srs.alloc();
    printf("code: %x\n", rw.getCode());
    rw.open(Resource::OPEN_READ_WRITE);

    ResourceOutputStream ros(&rw);
    for (int i = 0; i < 250; i++) {
        ros.write(i);
    }
    printf("rw size: %d\n", rw.size());
    ros.close();

    rw.open(Resource::OPEN_READ_ONLY);
    ResourceInputStream ris(&rw);
    while (ris.available()) {
        ris.read();
    }

    rw.close();
    _rs_io_memory_dump(srs.getRs());
    return 0;
}
```

5.5 Resource.h File Reference

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



Classes

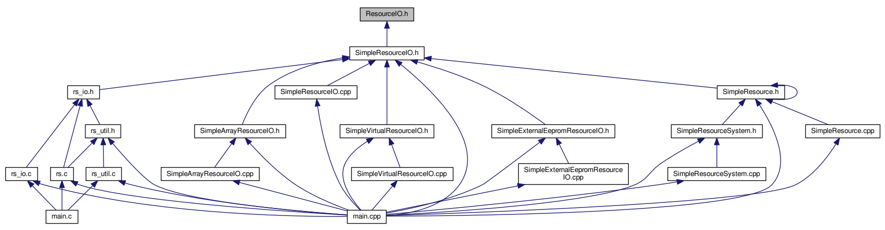
- class Resource

5.6 Resource.h

```

#ifndef __ARDUINO_RESOURCE_H__
#define __ARDUINO_RESOURCE_H__ 1

class Resource {
public:

    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_ALLOCATED = 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
    };

    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;
};

#endif // __ARDUINO_RESOURCE_H__
```

5.7 ResourceIO.h File Reference

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



Classes

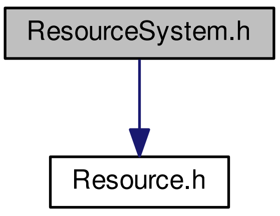
- class ResourceIO

5.8 ResourceIO.h

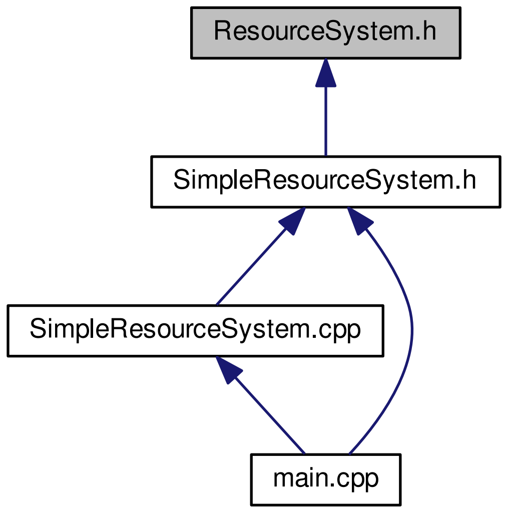
```
#ifndef __ARDUINO_RESOURCE_IO_H__
#define __ARDUINO_RESOURCE_IO_H__ 1

class ResourceIO {
public:

    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;
};

#endif /* __ARDUINO_RESOURCE_IO_H__ */

```

5.9 ResourceSystem.h File Reference

`#include <Resource.h>`
Include dependency graph for ResourceSystem.h:



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



Classes

- class ResourceSystem

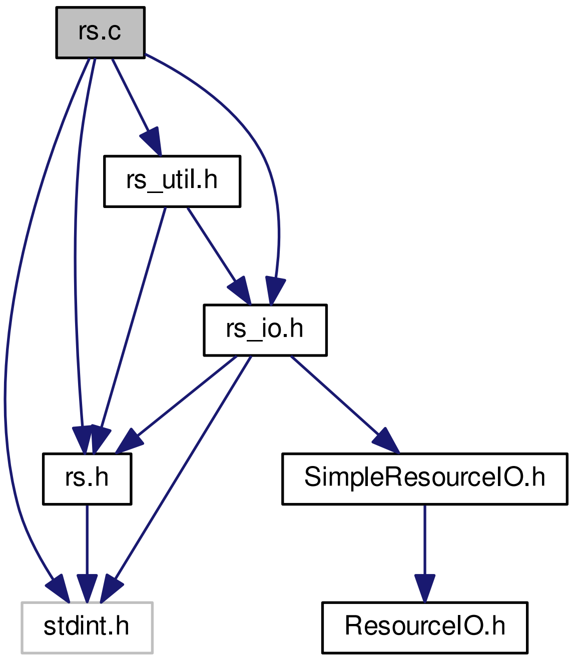
5.10 ResourceSystem.h

```

#ifndef __ARDUINO_RESOURCE_SYSTEM_H__
#define __ARDUINO_RESOURCE_SYSTEM_H__ 1

#include <Resource.h>

class ResourceSystem {
public:

    enum MountOptions {
        MOUNT_READ_WRITE = 0,
        MOUNT_READ_ONLY = 1
    };

    virtual bool mount(MountOptions options) = 0;

    virtual bool umount() = 0;

    virtual unsigned int totalSpace() = 0;

    virtual unsigned int availableSpace() = 0;
};

#endif /* __ARDUINO_RESOURCE_SYSTEM_H__ */
```

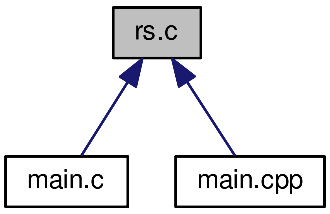
5.11 rs.c File Reference

```
#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_↩ 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

5.11.1 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

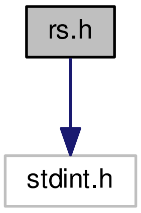
```
00001
00011 #ifndef __SDCC_RS_C__
00012 #define __SDCC_RS_C__ 1
00013
00014 #include "rs.h"
00015 #include "rs_io.h"
00016 #include "rs_util.h"
00017 #include <stdint.h>
00018
00019 rs_global_flags_t rs_global_flags;
00020
00021 rs_op_result_t rs_format(rs_t *rs) {
00022     uint8_t i;
00023    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

```c
    if ((options & RS_OPEN_RESOURCE_OPTION_READ_ONLY) || (rs->flags & RS_FLAG_BIT_READ_ONLY)) {
        flags |= RS_RESOURCE_FLAG_BIT_READ_ONLY;

    }
    _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(address), flags);
    resource->resource_descriptor = resource_descriptor;
    resource->first_cluster = _rs_io_read(rs->driver, RD_ADDRESS_TO_FIRST_CLUSTER(address));
    resource->current_cluster = resource->first_cluster;
    resource->cluster_offset = rs->sizeof_cluster_control;
    resource->current_position = 0;
    for (i = 0; i < RS_SIZEOF_RESOURCE_SIZE; i++) {
        *((uint8_t *) (&resource->size) + i) = _rs_io_read(rs->driver, address + i);
    }
    resource->flags = flags;
    _rs_check_for_eor_reached(resource);
    return RS_OP_RESULT_SUCCESS;
}

rs_op_result_t rs_close(rs_t *rs, rs_resource_t *resource) {
    rs_sync(rs, resource);
    _rs_free_resource_descriptor(rs, resource->resource_descriptor);
    resource->flags = ~RS_RESOURCE_FLAG_BIT_OPENED;
    return RS_OP_RESULT_SUCCESS;
}

uint8_t rs_read(rs_t *rs, rs_resource_t *resource) {
    rs_memory_address_t address;
    uint8_t read_data;
    if (!(resource->flags & RS_RESOURCE_FLAG_BIT_OPENED)) {
        resource->flags |= RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ;
        return 0;
    }
    if (_rs_is_eor_reached(resource)) {
        return 0;
    }
    _rs_check_for_availability(rs, resource);
    address = _rs_cluster_to_address(rs, resource->current_cluster);
    read_data = _rs_io_read(rs->driver, address + resource->cluster_offset);
    resource->current_position++;
    resource->cluster_offset++;
    _rs_check_for_eor_reached(resource);
    return read_data;
}

rs_op_result_t rs_write(rs_t *rs, rs_resource_t *resource, uint8_t data_to_write) {
    rs_memory_address_t address;
    if (!(resource->flags & RS_RESOURCE_FLAG_BIT_OPENED)) {
        return RS_OP_RESULT_ERROR_RESOURCE_CLOSED;
    }
    if (resource->flags & RS_RESOURCE_FLAG_BIT_READ_ONLY) {
        return RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY;
    }
    if (!_rs_check_for_availability(rs, resource)) {
        return RS_OP_RESULT_ERROR_NO_SPACE_AVAILABLE;
    }
    address = _rs_cluster_to_address(rs, resource->current_cluster);
    _rs_io_write(rs->driver, address + resource->cluster_offset, data_to_write);
    resource->cluster_offset++;
    resource->current_position++;
    if (rs_eor(resource)) {
        resource->size++;

        /*
         * It is causing seriously performance problems. Since the IO has a
         * buffer, and we are writing at this buffer, once we need to sync, this
         * buffer will be flushed to the resource and the IO pointer will need
         * to go to the resource descriptor to write the new size of the
         * resource. After the sync, we will continue to write the next byte,
         * the buffer filled on the sync will be lost and a new buffer will be
         * created to write the next byte... and so on.
         */
        //rs_sync(rs, resource);
    }
    return RS_OP_RESULT_SUCCESS;
}

rs_op_result_t rs_seek(rs_t *rs, rs_resource_t *resource,
rs_seek_origin_t origin, rs_seek_int_t offset) {
    int16_t new_position = 0;
    if (resource->size == 0) {
        return RS_OP_RESULT_SUCCESS;
    }
    switch (origin) {
        case RS_SEEK_ORIGIN_BEGIN:
            new_position = offset;
            break;
        case RS_SEEK_ORIGIN_CURRENT:
            new_position = resource->current_position + offset;
            break;
    }
    new_position %= resource->size + 1;
    if (new_position < 0) {
        new_position += resource->size;
    }
    if (new_position == 0) {
        rs_rewind(rs, resource);
        return RS_OP_RESULT_SUCCESS;
    }
    if (new_position < resource->current_position) {
        if (new_position > (resource->current_position - new_position)) {
            _rs_move_current_position_back(rs, resource, (resource->current_position - new_position));
        } else {
            rs_rewind(rs, resource);
            _rs_move_current_position_ahead(rs, resource, new_position);
        }
    } else {
        _rs_move_current_position_ahead(rs, resource, (new_position - resource->current_position));
    }
    _rs_check_for_eor_reached(resource);
    return RS_OP_RESULT_SUCCESS;
}

rs_op_result_t rs_truncate(rs_t *rs, rs_resource_t *resource) {
    uint8_t flags;
    rs_memory_address_t resource_descriptor_address;
    uint8_t freed_clusters = 0;
    resource_descriptor_address = _rs_resource_descriptor_to_address(rs, resource->resource_descriptor);
    flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(resource_descriptor_address));
    if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
        return RS_OP_RESULT_ERROR_RESOURCE_DOES_NOT_ALLOCATED;
    }
    if (resource->size > rs->sizeof_cluster_data) {
        freed_clusters = _rs_format_clusters_chain(rs, _rs_next_cluster_by_cluster(rs, resource->first_cluster));
    }
    _rs_increase_free_clusters(rs, freed_clusters);
    resource->size = 0x00;
    _rs_io_write(rs->driver, RD_ADDRESS_TO_SIZE_LOW(resource_descriptor_address), 0x00);
    _rs_io_write(rs->driver, RD_ADDRESS_TO_SIZE_HIGH(resource_descriptor_address), 0x00);
    return RS_OP_RESULT_SUCCESS;
}

void rs_sync(rs_t *rs, rs_resource_t *resource) {
    uint8_t i;
    rs_memory_address_t address;
    address = _rs_resource_descriptor_to_address(rs, resource->resource_descriptor);
    for (i = 0; i < 2; i++) {
        _rs_io_write(rs->driver, address + i, *((uint8_t *) (&(resource->size)) + i));
    }
}

void rs_stat(rs_t *rs, rs_resource_t *resource, rs_stat_t *stat) { // TODO
    stat->flags = 0xff;
}

rs_op_result_t rs_rewind(rs_t *rs, rs_resource_t *resource) {
    resource->current_cluster = resource->first_cluster;
    resource->cluster_offset = rs->sizeof_cluster_control;
    resource->current_position = 0;
    _rs_check_for_eor_reached(resource);
    return RS_OP_RESULT_SUCCESS;
}

```

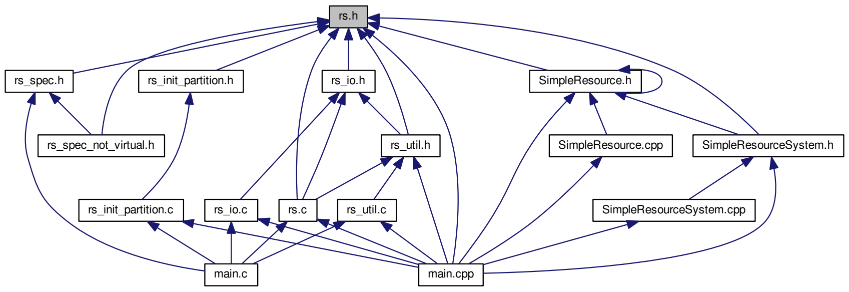
```
rs_resource_code_t rs_alloc(rs_t *rs) {
    uint8_t i;
    uint8_t flags;
    rs_cluster_t first_cluster;
    rs_memory_address_t resource_descriptor_address, cluster_address;
    if (rs->free_clusters < 1) {
        return RS_NULL_RESOURCE_CODE;
    }
    resource_descriptor_address = rs->resource_descriptor_table_address;
    for (i = 0; i < rs->resource_descriptor_count; i++) {
        flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(resource_descriptor_address));
        if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
            cluster_address = _rs_alloc_cluster(rs);
            if (cluster_address == RS_NULL_CLUSTER_ADDRESS) {
                return RS_NULL_RESOURCE_CODE;
            }
            flags |= RS_RESOURCE_FLAG_BIT_ALLOCATED;
            first_cluster = _rs_address_to_cluster(rs, cluster_address);
            _rs_create_cluster_chain(rs, first_cluster, RS_INEXISTENT_CLUSTER);
            _rs_io_write(rs->driver, RD_ADDRESS_TO_FIRST_CLUSTER(resource_descriptor_address), first_cluster);
            _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(resource_descriptor_address), flags);
            return _rs_resource_descriptor_to_resource_code(i);
        }
        resource_descriptor_address += rs->sizeof_resource_descriptor;
    }
    return RS_NULL_RESOURCE_CODE;
}

uint8_t rs_release(rs_t *rs, rs_resource_t *resource) {
    uint8_t flags;
    rs_memory_address_t resource_descriptor_address;
    resource_descriptor_address = _rs_resource_descriptor_to_address(rs, resource->resource_descriptor);
    flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(resource_descriptor_address));
    if (!(flags & RS_RESOURCE_FLAG_BIT_ALLOCATED)) {
        return 1;
    }
    _rs_format_resource_clusters(rs, resource);
    _rs_format_resorce_descriptor(rs, resource->resource_descriptor);
    resource->flags = 0x00;
    return 1;
}

rs_resource_size_t rs_size(rs_resource_t *resource) {
    return resource->size;
}

rs_resource_size_t rs_tell(rs_resource_t *resource) {
    return resource->current_position;
}

uint8_t rs_eor(rs_resource_t *resource) {
    return _rs_is_eor_reached(resource);
}

uint8_t rs_error(rs_resource_t *resource) {
    return (resource->flags & RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ || resource->flags & RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE);
}

rs_resource_size_t rs_available_space(rs_t *rs) {
    return rs->free_clusters * rs->sizeof_cluster_data;
}

rs_resource_size_t rs_total_space(rs_t *rs) {
    return rs->cluster_count * rs->sizeof_cluster_data;
}

#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_RESORCE_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_RESORCE_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_umount (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.

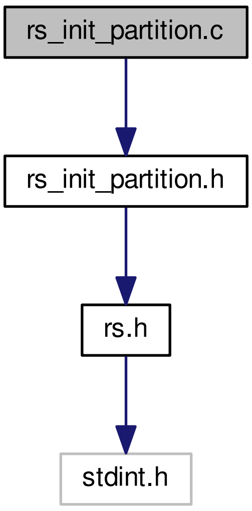
5.14 rs.h

```

#ifndef __SDCC_RS_H__
#define __SDCC_RS_H__ 1

#include <stdint.h>

#define RS_NULL_RESOURCE_CODE                                       0xff
#define RS_NULL_CLUSTER                                             0xff

#define RS_NULL_RESORCE_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)

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;

// Drivers

typedef enum {
    RS_DRIVER_VIRTUAL = 0,
    RS_DRIVER_SELF_EEPROM = 1,
    RS_DRIVER_MULTI_EXTERNAL_EEPROM = 2,
    RS_DRIVER_EXTERNAL_EEPROM = 3,
    RS_DRIVER_ARDUINO_EEPROM = 4
} rs_driver_t;

// Resource fag bit values

typedef enum {
    RS_RESOURCE_FLAG_BIT_OPENED = 1,
    RS_RESOURCE_FLAG_BIT_READ_ONLY = 2,
    RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ = 4,
    RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_WRITE = 8,
    RS_RESOURCE_FLAG_BIT_ALLOCATED = 16,
    RS_RESOURCE_FLAG_BIT_EOR_REACHED = 32
} rs_resource_flag_bits_t;

// Options to open a resource

typedef enum {
    RS_OPEN_RESOURCE_OPTION_NORMAL = 0,
    RS_OPEN_RESOURCE_OPTION_READ_ONLY = 1
} rs_open_resource_options_t;

// Options to mount a resource

typedef enum {
    RS_MOUNT_OPTION_NORMAL = 0,
    RS_MOUNT_OPTION_READ_ONLY = 1
} rs_mount_options_t;

// Rs fag bit values

typedef enum {
    RS_FLAG_BIT_DRIVER_MOUNTED = 1,
    RS_FLAG_BIT_READ_ONLY = 2
} rs_flag_bits_t;

// Operation result

typedef enum {
    RS_OP_RESULT_SUCCESS = 0,
    RS_OP_RESULT_ERROR_RESOURCE_OPENED = 1,
    RS_OP_RESULT_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_DOES_NOT_ALLOCATED = 7,
    RS_OP_RESULT_ERROR_DRIVER_NOT_MOUNTED = 8
} rs_op_result_t;

// Seek position reference

typedef enum {
    RS_SEEK_ORIGIN_BEGIN = 0,
    RS_SEEK_ORIGIN_CURRENT = 1
} rs_seek_origin_t;

typedef struct {
    uint8_t flags;
} rs_stat_t;

// Resource system

typedef struct {
    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;
} rs_t;

// Resource

typedef struct {
    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;
} rs_resource_t;

typedef struct {
    uint8_t driver_mouted;
} rs_global_flags_t;

extern rs_global_flags_t rs_global_flags;

// Format a device
rs_op_result_t rs_format(rs_t *rs);

// Register a work area
rs_op_result_t rs_mount(rs_driver_t driver,
rs_t *rs, rs_mount_options_t options);

// Unregister a work area
rs_op_result_t rs_umount(rs_t *rs);

// Open/Create a resource (you must give a empty resource)
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);

// Close a resource
rs_op_result_t rs_close(rs_t *rs, rs_resource_t *resource);

// Read a byte from resource
uint8_t rs_read(rs_t *rs, rs_resource_t *resource);

// Write a byte from resource
rs_op_result_t rs_write(rs_t *rs, rs_resource_t *resource, uint8_t
data_to_write);

// Move read/write pointer, (Expand resource size not implemented yet)
rs_op_result_t rs_seek(rs_t *rs, rs_resource_t *resource,
rs_seek_origin_t origin, rs_seek_int_t offset);

// Truncate resource size
rs_op_result_t rs_truncate(rs_t *rs, rs_resource_t *resource);

// Flush cached data
void rs_sync(rs_t *rs, rs_resource_t *resource);

// Get descriptor status
void rs_stat(rs_t *rs, rs_resource_t *resource,
rs_stat_t *stat);

// Rewind the position of a resource pointer
rs_op_result_t rs_rewind(rs_t *rs, rs_resource_t *resource);

// Create/Allocate a new resource if available
rs_resource_code_t rs_alloc(rs_t *rs);

// Make a resource free to be allocated for another one
uint8_t rs_release(rs_t *rs, rs_resource_t *resource);

// Get size of a resource
rs_resource_size_t rs_size(rs_resource_t *resource);

// Get the current read/write pointer
rs_resource_size_t rs_tell(rs_resource_t *resource);

// Test for end-of-resource on a resource
uint8_t rs_eor(rs_resource_t *resource);

// Test for an error on a resource
uint8_t rs_error(rs_resource_t *resource);

// Return the current available space in the partition
rs_resource_size_t rs_available_space(rs_t *rs);

// Return the total space in the partition
```

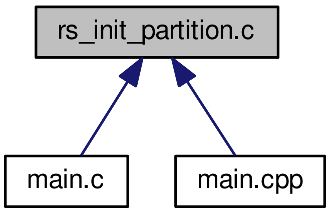
```
rs_resource_size_t rs_total_space(rs_t *rs);

#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

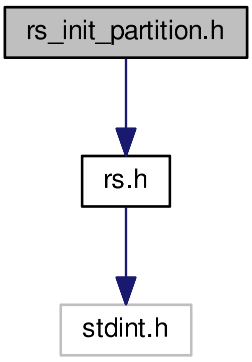
```
00001
00011 #ifndef __SDCC_RS_INIT_PARTITION_C__
00012 #define __SDCC_RS_INIT_PARTITION_C__ 1
00013
00014 #include "rs_init_partition.h"
00015
00016 void rs_init_partition(rs_t *rs, rs_disk_size_t size,
      rs_environment_t env) {
00017
00018     switch(size) {
00019
00020         case RS_DISK_32K:
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;
```

```
        break;

        case RS_DISK_8K:
            if (env == RS_ENV_VIRTUAL) {
                rs->driver = RS_DRIVER_VIRTUAL;
            } else {
                rs->driver = RS_DRIVER_ARDUINO_EEPROM;
            }
            rs->memory_size = 0x2000; //8192;
            rs->resource_descriptor_table_address = 0x0020; //32;
            rs->cluster_table_address = 0x00a0; //160;
            rs->sizeof_resource_descriptor_table = 0x0080; //128;
            rs->sizeof_cluster_table = 0x1f60; //8032;
            rs->sizeof_resource_descriptor = 0x04; //4;
            rs->sizeof_cluster = 0x20; //32;
            rs->resource_descriptor_count = 0x20; //32;
            rs->cluster_count = 0xfb; //251;
            rs->sizeof_cluster_data = 0x1e; //30;
            rs->sizeof_cluster_control = 0x02; //2;
            rs->free_clusters = 0xfb; //251;
            rs->flags = 0x00; //0;
        break;

        default:

            if (env == RS_ENV_VIRTUAL) {
                rs->driver = RS_DRIVER_VIRTUAL;
            } else {
                rs->driver = RS_DRIVER_ARDUINO_EEPROM;
            }
            rs->memory_size = 0xf46; //3910;
            rs->resource_descriptor_table_address = 0x0020; //32;
            rs->cluster_table_address = 0x00a0; //160;
            rs->sizeof_resource_descriptor_table = 0x0080; //128;
            rs->sizeof_cluster_table = 0xea6; //3750;
            rs->sizeof_resource_descriptor = 0x04; //4;
            rs->sizeof_cluster = 0x0f; //32;
            rs->resource_descriptor_count = 0x20; //32;
            rs->cluster_count = 0xfa; //250;
            rs->sizeof_cluster_data = 0x0d; //13;
            rs->sizeof_cluster_control = 0x02; //2;
            rs->free_clusters = 0xfa; //250;
            rs->flags = 0x00; //0;
        break;
    }
}

#endif // __SDCC_RS_INIT_PARTITION_C__
```

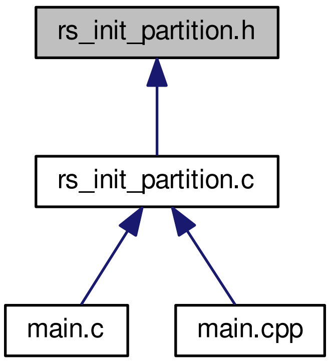
5.17 rs_init_partition.h File Reference

`#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*)

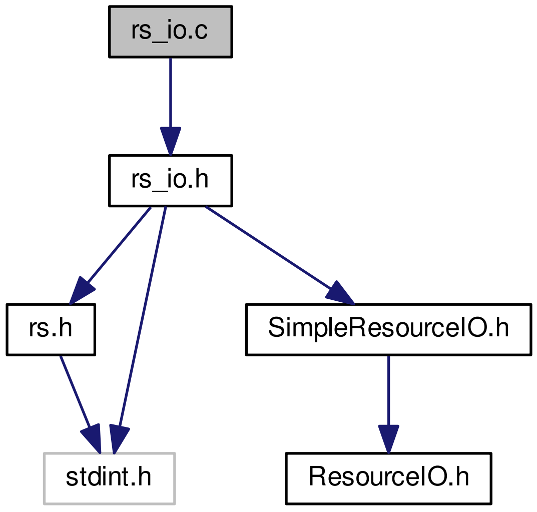
Definition at line 16 of file rs_init_partition.c.

5.18 rs_init_partition.h

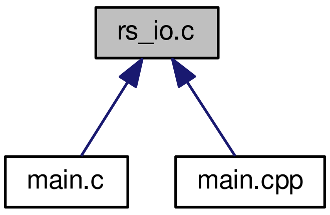
```
00001
00011 #ifndef __SDCC_RS_INIT_PARTITION_H__
00012 #define __SDCC_RS_INIT_PARTITION_H__ 1
00013
00014 #include "rs.h"
00015
00016 typedef enum {
00017     RS_DISK_4K,
00018     RS_DISK_8K,
00019     RS_DISK_24K,
00020     RS_DISK_32K
00021 } rs_disk_size_t;
00022
00023 typedef enum {
00024     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,
      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.

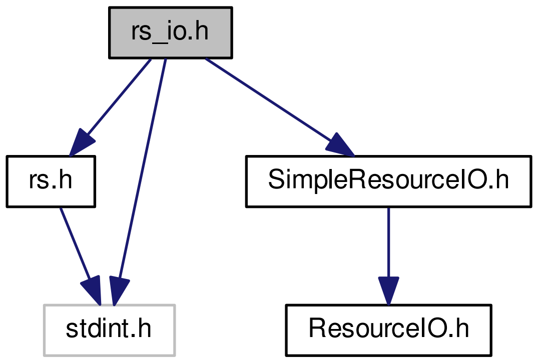
5.20 rs_io.c

```
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,
      rs_memory_address_t address) {
00017     return SimpleResourceIO::getAssociatedIO(driver)->
      read(address);
00018 }
00019
00020 void _rs_io_write(rs_driver_t driver, rs_memory_address_t address
      , uint8_t data) {
00021     SimpleResourceIO::getAssociatedIO(driver)->
      write(address, data);
00022 }
00023
00024 #endif // __SDCC_RS_IO_C__
```

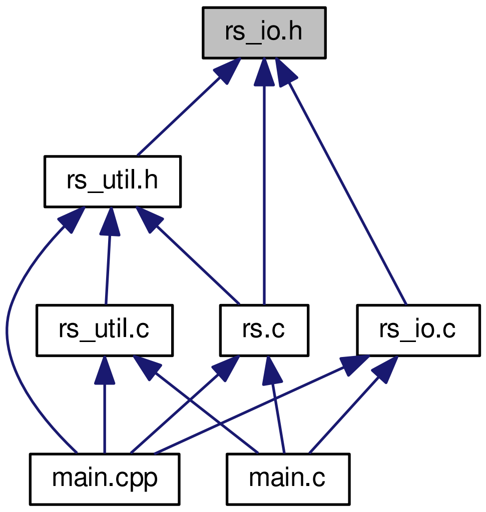
5.21 rs_io.h File Reference

```
#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.

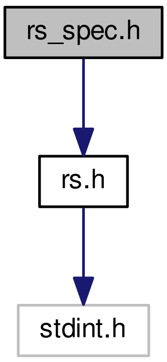
5.22 rs_io.h

```

#ifndef __SDCC_RS_IO_H__
#define __SDCC_RS_IO_H__ 1

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

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);

#endif // __SDCC_RS_IO_H__
```

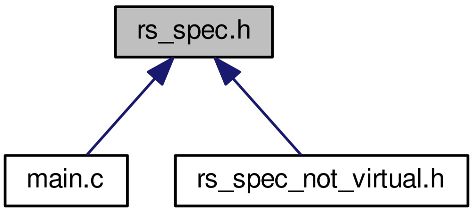
5.23 rs_spec.h File Reference

`#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

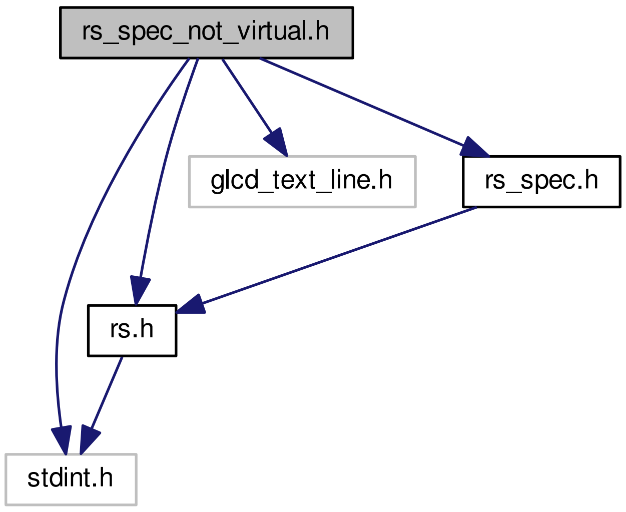
```
#include "rs.h"

#if VIRTUAL_ENVIROMENT == 1
#define RS_SPEC_DRIVER RS_DRIVER_VIRTUAL
#else
#define RS_SPEC_DRIVER RS_DRIVER_EXTERNAL_EEPROM
#endif

#ifndef rs_spec_printf
#define rs_spec_printf printf
#endif

#ifndef RS_SPEC_IGNORE_0

void format_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) fomat spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) fomat spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
}
#endif

#ifndef RS_SPEC_IGNORE_1

void mount_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) mount spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) mount spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_2

void umount_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    op_r = rs_umount(rs);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) umount spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) umount spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
}
#endif

#ifndef RS_SPEC_IGNORE_3

void alloc_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
        rs_spec_printf("(F) alloc_resource spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) alloc_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_4

void try_to_alloc_resources_that_is_possible_spec(
rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code[2];
    uint8_t i = 0;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    for (; i < rs->resource_descriptor_count; i++) {
        rs_resource_code[0] = rs_alloc(rs);
    }
    rs_resource_code[1] = rs_alloc(rs);
    if (rs_resource_code[0] == (rs->resource_descriptor_count - 1) &&
rs_resource_code[1] == RS_NULL_RESOURCE_CODE) {
        rs_spec_printf("(*) try_to_alloc_resources_that_is_possible spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) try_to_alloc_resources_that_is_possible spec failed. error: %x\n"
, rs_resource_code[0]);
        while (1);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_5

void open_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) open_resource spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) open_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_6

void write_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_write(rs, &resource, 0xaa);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) write_resource spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) write_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_7

void rewind_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_write(rs, &resource, 0xAA);
    op_r = rs_rewind(rs, &resource);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) rewind_resource spec failed. error: %d\n", op_r);
    } else {
        rs_spec_printf("(*) rewind_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_8

void read_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    unsigned char c[2];
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_write(rs, &resource, 0x41);
```

```c
    op_r = rs_write(rs, &resource, 0xA1);
    op_r = rs_rewind(rs, &resource);
    c[0] = rs_read(rs, &resource);
    c[1] = rs_read(rs, &resource);
    if (c[0] != 0x41 || c[1] != 0xA1) {
        rs_spec_printf("(F) read_resource spec failed. error: %x\n", c[0]);
    } else {
        rs_spec_printf("(*) read_resource spec passed.\n", RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_9

void close_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    unsigned char c[2];
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_close(rs, &resource);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(F) close_resource spec failed. error: %x\n", op_r);
    } else {
        rs_spec_printf("(*) close_resource spec passed.\n", RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_10

void try_read_when_end_of_resource_is_reached_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_write(rs, &resource, 0x41);
    op_r = rs_rewind(rs, &resource);
    rs_read(rs, &resource);
    rs_read(rs, &resource);
    if (op_r == 0 && (rs_eor(&resource))) {
        rs_spec_printf("(*) try_read_when_end_of_resource_is_reached spec passed.\n", RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) try_read_when_end_of_resource_is_reached spec failed. error: %x\n", op_r);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_11

void try_read_when_resource_is_closed_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_close(rs, &resource);
    rs_read(rs, &resource);
    if (op_r == 0 && (resource.flags |
RS_RESOURCE_FLAG_BIT_ERROR_ON_LAST_READ)) {
        rs_spec_printf("(*) try_read_when_resource_is_closed spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) try_read_when_resource_is_closed spec failed. error: %x\n", op_r)
;
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_12

void seek_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < 50; i++) {
        op_r = rs_write(rs, &resource, (i + 0x65));
    }
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
    if (op_r == RS_OP_RESULT_SUCCESS) {
        rs_spec_printf("(*) seek_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) seek_resource spec failed. error: %x\n", op_r);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_13

void random_read_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    unsigned char c[5], first_write_char = 0x65;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < 255; i++) {
        op_r = rs_write(rs, &resource, (i + first_write_char));
    }
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
    c[0] = rs_read(rs, &resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 10);
    c[1] = rs_read(rs, &resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 48);
    c[2] = rs_read(rs, &resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 20);
    c[3] = rs_read(rs, &resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 0);
    c[4] = rs_read(rs, &resource);
    if (c[0] == first_write_char + 20 &&
            c[1] == first_write_char + 31 &&
            c[2] == first_write_char + 48 &&
            c[3] == first_write_char + 69 &&
            c[4] == first_write_char + 0) {
        rs_spec_printf("(*) random_read_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) random_read_resource spec failed. error: %x\n", op_r);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_14

void random_read_with_seek_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    unsigned char c[255];
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (i = 0; i < 255; i++) {
        op_r = rs_write(rs, &resource, i);
    }
    rs_rewind(rs, &resource);

    for (i = 0; i < 255; i++) {
        rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, i);
        c[i] = rs_read(rs, &resource);
        rs_read(rs, &resource);
        rs_read(rs, &resource);
        rs_read(rs, &resource);
    }

    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 199);
    if ((i = rs_read(rs, &resource)) != 199) {
        rs_spec_printf("(F) random_read_with_seek_resource_spec spec failed. != 199\n", 0);
    }

    for (i = 0; i < 255; i++) {
        if (i != c[i]) {
            rs_spec_printf("(F) random_read_with_seek_resource_spec spec failed. error: %x\n"
, i);
        }
    }

    rs_spec_printf("(*) random_read_with_seek_resource_spec spec passed.\n",
RS_OP_RESULT_SUCCESS);
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_15

void random_read_with_seek_opening_resource_spec(
rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    unsigned char c[255];
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (i = 0; i < 255; i++) {
        op_r = rs_write(rs, &resource, i);
    }
    rs_close(rs, &resource);

    for (i = 0; i < 255; i++) {
        op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
        rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, i);
        c[i] = rs_read(rs, &resource);
        rs_read(rs, &resource);
        rs_read(rs, &resource);
        rs_read(rs, &resource);
        rs_close(rs, &resource);
    }

    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 199);
    if ((i = rs_read(rs, &resource)) != 199) {
        rs_spec_printf("(F) random_read_with_seek_opening_resource_spec spec failed. != 199\n
", op_r);
    }
    rs_close(rs, &resource);

    for (i = 0; i < 255; i++) {
        if (i != c[i]) {
            rs_spec_printf("(F) random_read_with_seek_opening_resource_spec spec failed. error: %x\n", c[i]);
        }
    }

    rs_spec_printf("(*) random_read_with_seek_opening_resource_spec spec passed.\n", RS_OP_RESULT_SUCCESS);
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_16

void size_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint16_t i = 0;
    uint16_t size = 0xf40;
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < size; i++) {
        op_r = rs_write(rs, &resource, 0x65);
    }
    if (rs_size(&resource) == 0xf40) {
        rs_spec_printf("(*) size_resource spec passed.\n", RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) size_resource spec failed. error: %x\n", size);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_17

void tell_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < 50; i++) {
        op_r = rs_write(rs, &resource, 0x65);
    }
    if (rs_tell(&resource) == 50) {
        rs_spec_printf("(*) tell_resource spec passed.\n", RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) tell_resource spec failed. error: %x\n", op_r);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_18

void tell_with_seek_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    uint8_t i = 0;
    rs_resource_size_t s[5];
    rs_init_partition(rs, RS_DISK_32K, RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs, RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource, RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < 50; i++) {
        op_r = rs_write(rs, &resource, 0x65);
    }
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 20);
    s[0] = rs_tell(&resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 10);
    s[1] = rs_tell(&resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 48);
    s[2] = rs_tell(&resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_CURRENT, 20);
    s[3] = rs_tell(&resource);
    op_r = rs_seek(rs, &resource, RS_SEEK_ORIGIN_BEGIN, 0);
    s[4] = rs_tell(&resource);
    if (s[0] == 20 &&
            s[1] == 30 &&
            s[2] == 48 &&
            s[3] == 17 &&
            s[4] == 0) {
        rs_spec_printf("(*) tell_with_seek_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) tell_with_seek_resource spec failed. error: %d\n", s[3]);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_19

void total_space_resource_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_code_t rs_resource_code;
    rs_resource_t resource;
    rs_resource_size_t total_space[2];
    uint16_t i = 0;
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    total_space[0] = rs_available_space(rs);
    rs_resource_code = rs_alloc(rs);
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    for (; i < rs->sizeof_cluster_data + 1; i++) {
        op_r = rs_write(rs, &resource, 0x65);
    }
    total_space[1] = rs_available_space(rs);
    if (total_space[0] - total_space[1] == (rs->sizeof_cluster_data * 2)) {
        rs_spec_printf("(*) total_space_resource spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf("(F) total_space_resource spec failed. error: %d != 50\n", total_space
[0] - total_space[1]);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_20

void allocating_multi_format_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_t resource;
    uint8_t count = 3;
    uint8_t j, i;
    rs_resource_code_t rs_resource_code[3];
    uint8_t passed = 1;
    for (j = 0; j < count; j++) {
        rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
        op_r = rs_format(rs);
        op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
        rs_resource_code[j] = rs_alloc(rs);
        op_r = rs_open(rs, rs_resource_code[j], &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
        for (i = 0; i < 50; i++) {
            op_r = rs_write(rs, &resource, 0x65);
        }
        rs_close(rs, &resource);
    }
    for (j = 0; j < count; j++) {
        if (rs_resource_code[j] != 0) {
            rs_spec_printf("(F) allocating_multi_format spec failed %x\n", rs_resource_code[j
]);
            passed = 0;
        }
    }
    if (passed) {
        rs_spec_printf("(*) allocating_multi_format spec passed\n",
RS_OP_RESULT_SUCCESS);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_21

void read_only_mounting_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_t resource;
    rs_resource_code_t rs_resource_code;
    char error_msg[] = "(F) read_only_mounting spec failed. %d\n";
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_READ_ONLY);
    rs_resource_code = rs_alloc(rs);
    if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
        rs_spec_printf(error_msg, op_r);
    }
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_NORMAL);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf(error_msg, op_r);
    }
    op_r = rs_write(rs, &resource, 0xaa);
    if (op_r == RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
        rs_spec_printf("(*) read_only_mounting spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf(error_msg, op_r);
    }
    rs_umount(rs);
}
#endif

#ifndef RS_SPEC_IGNORE_22

void read_only_opening_spec(rs_t *rs) {
    rs_op_result_t op_r;
    rs_resource_t resource;
    rs_resource_code_t rs_resource_code;
    char error_msg[] = "(F) read_only_opening spec failed. %d\n";
    rs_init_partition(rs, RS_DISK_32K,
RS_ENV_VIRTUAL);
    op_r = rs_format(rs);
    op_r = rs_mount(RS_SPEC_DRIVER, rs,
RS_MOUNT_OPTION_NORMAL);
    rs_resource_code = rs_alloc(rs);
    if (rs_resource_code == RS_NULL_RESOURCE_CODE) {
        rs_spec_printf(error_msg, op_r);
    }
    op_r = rs_open(rs, rs_resource_code, &resource,
RS_OPEN_RESOURCE_OPTION_READ_ONLY);
    if (op_r != RS_OP_RESULT_SUCCESS) {
        rs_spec_printf(error_msg, op_r);
    }
    op_r = rs_write(rs, &resource, 0xaa);
    if (op_r == RS_OP_RESULT_ERROR_RESOURCE_READ_ONLY) {
        rs_spec_printf("(*) read_only_opening spec passed.\n",
RS_OP_RESULT_SUCCESS);
    } else {
        rs_spec_printf(error_msg, op_r);
    }
    rs_umount(rs);
}
#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

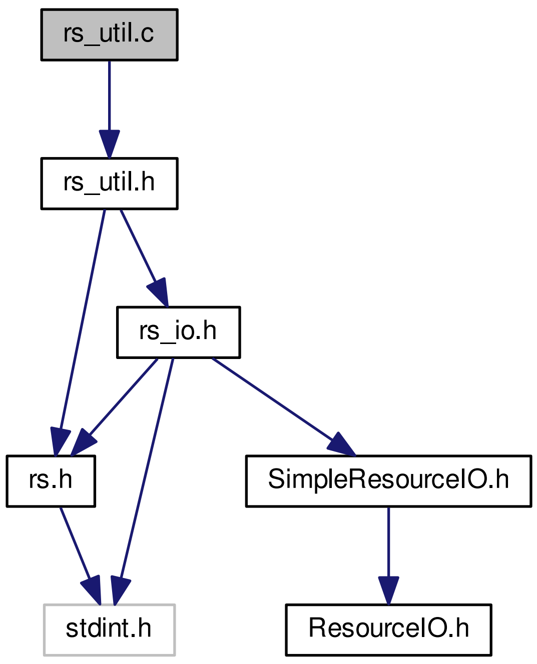
```
00001 #include <stdint.h>
00002 #include "rs.h"
00003
00004 #include <glcd_text_line.h>
00005
00006 void print_ln(char *s, uint8_t r) {
00007     char b[3];
00008     i8toh(r, b);
00009     glcd_text_line_print_ln(s);
00010     glcd_text_line_print_ln(b);
00011 }
00012
00013 #define rs_spec_printf 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
```

```
#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

#include "rs_spec.h"
```

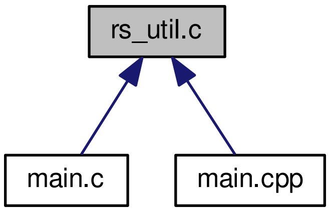
5.27 rs_util.c File Reference

`#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

rs	

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

rs	
resource	

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 respecitve flag.

Parameters

resource	

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

rs	
prev_cluster	
next_cluster	

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

rs	

cluster	

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

rs	
cluster	

Returns

Definition at line 196 of file rs_util.c.

5.27.2.7 void _rs_format_resorce_descriptor (**rs_t** ∗ *rs,* **rs_resource_descriptor_t** *resource_descriptor*)

Free a resource description.

Parameters

rs	
resource_↩ descriptor	

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

rs	
resource	

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

rs	
cluster	

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

rs	
resource_↩ descriptor	

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

rs	

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

rs	

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

driver	

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

resource	

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

rs	
cluster	

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

rs	
resource	
offset	

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

rs	
resource	
offset	

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

driver	
rs	

Definition at line 26 of file rs_util.c.

5.27.2.19 void _rs_set_driver_monted (rs_driver_t *driver*, uint8_t *is*)

Set/clear given driver as mouted.

Parameters

driver	
is	

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

driver	
rs	

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     }
}

void _rs_check_for_eor_reached(rs_resource_t *resource) {
    if (resource->current_position >= resource->size) {
        resource->flags |= RS_RESOURCE_FLAG_BIT_EOR_REACHED;
    } else {
        resource->flags &= ~RS_RESOURCE_FLAG_BIT_EOR_REACHED;
    }
}

uint8_t _rs_is_eor_reached(rs_resource_t *resource) {
    return resource->flags & RS_RESOURCE_FLAG_BIT_EOR_REACHED;
}

uint8_t _rs_check_for_availability(rs_t *rs, rs_resource_t *resource) {
    rs_memory_address_t address;
    rs_cluster_t cluster;
    _rs_check_for_eor_reached(resource);
    if (resource->cluster_offset >= rs->sizeof_cluster) {
        if (rs_eor(resource)) {
            address = _rs_alloc_cluster(rs);
            if (address == RS_NULL_CLUSTER_ADDRESS) {
                return 0;
            }
            cluster = _rs_address_to_cluster(rs, address);
            _rs_create_cluster_chain(rs, resource->current_cluster, cluster);
            resource->current_cluster = cluster;
        } else {
            resource->current_cluster = _rs_next_cluster_by_cluster(rs, resource->current_cluster);
        }
        resource->cluster_offset = rs->sizeof_cluster_control;
    }
    return 1;
}

uint8_t _rs_move_current_position_ahead(rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset) {
    uint8_t until_the_end;
    uint8_t how_many_clustes_ahead;
    uint8_t i;
    resource->current_position += offset;
    until_the_end = (rs->sizeof_cluster - resource->cluster_offset);
    if (offset <= until_the_end) {
        resource->cluster_offset += offset;
        return 1;
    }
    offset -= until_the_end;
    how_many_clustes_ahead = (offset / rs->sizeof_cluster_data) + 1;
    resource->cluster_offset = (offset % rs->sizeof_cluster_data) + rs->sizeof_cluster_control;
    for (i = 0; i < how_many_clustes_ahead; i++) {
        resource->current_cluster = _rs_next_cluster_by_cluster(rs, resource->current_cluster);
    }
    return 1;
}

uint8_t _rs_move_current_position_back(rs_t *rs, rs_resource_t *resource, rs_seek_int_t offset) {
    uint8_t until_the_begin;
    uint8_t how_many_clustes_back;
    uint8_t i;
    resource->current_position -= offset;
    until_the_begin = (resource->cluster_offset - rs->sizeof_cluster_control);
    if (offset <= until_the_begin) {
        resource->cluster_offset -= offset;
        return 1;
    }
    offset -= until_the_begin;
    how_many_clustes_back = (offset / rs->sizeof_cluster_data);
    if ((offset % rs->sizeof_cluster_data) != 0) {
        how_many_clustes_back++;
    }
    resource->cluster_offset = rs->sizeof_cluster - (offset % rs->sizeof_cluster_data);
    for (i = 0; i < how_many_clustes_back; i++) {
        resource->current_cluster = _rs_prev_cluster_by_cluster(rs, resource->current_cluster);
    }
    return 1;
}

```

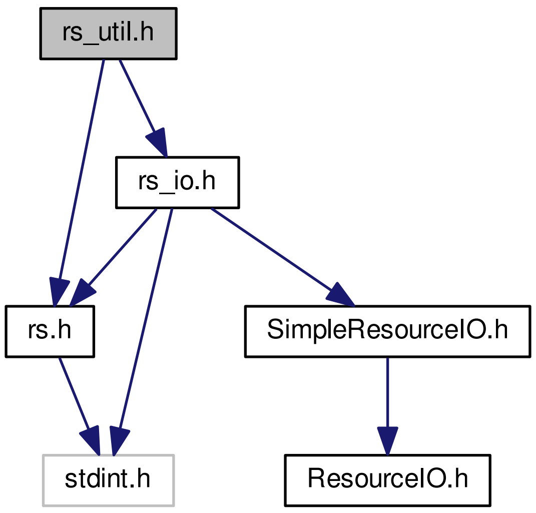
```c
void _rs_format_resorce_descriptor(rs_t *rs, rs_resource_descriptor_t resource_descriptor) {
    int i;
    rs_memory_address_t address;
    address = _rs_resource_descriptor_to_address(rs, resource_descriptor);
    for (i = 0; i < rs->sizeof_resource_descriptor; i++) {
        _rs_io_write(rs->driver, address + i, 0x00);
    }
}

uint8_t _rs_is_driver_monted(rs_driver_t driver) {
    return rs_global_flags.driver_mouted & (1 << driver);
}

void _rs_set_driver_monted(rs_driver_t driver, uint8_t is) {
    if (is) {
        rs_global_flags.driver_mouted |= (1 << driver);
    } else {
        rs_global_flags.driver_mouted &= ~(1 << driver);
    }
}

void _rs_free_resource_descriptors(rs_t *rs) {
    uint8_t i;
    for (i = 0; i < rs->resource_descriptor_count; i++) {
        _rs_free_resource_descriptor(rs, i);
    }
}

void _rs_free_resource_descriptor(rs_t *rs, rs_resource_descriptor_t resource_descriptor) {
    rs_memory_address_t address;
    uint8_t flags;
    address = _rs_resource_descriptor_to_address(rs, resource_descriptor);
    flags = _rs_io_read(rs->driver, RD_ADDRESS_TO_FLAG(address));
    flags &= ~(RS_RESOURCE_FLAG_BIT_OPENED | RS_RESOURCE_FLAG_BIT_READ_ONLY);
    _rs_io_write(rs->driver, RD_ADDRESS_TO_FLAG(address), flags);
}

void _rs_format_resource_clusters(rs_t *rs, rs_resource_t *resource) {
    uint8_t freed_clusters;
    freed_clusters = _rs_format_clusters_chain(rs, resource->first_cluster);
    _rs_increase_free_clusters(rs, freed_clusters);
}

uint8_t _rs_format_clusters_chain(rs_t *rs, rs_cluster_t cluster) {
    rs_cluster_t next_cluster;
    uint8_t formated_clusters = 0;
    do {
        next_cluster = _rs_next_cluster_by_cluster(rs, cluster);
        _rs_format_cluster(rs, cluster);
        formated_clusters++;
        if (next_cluster == RS_INEXISTENT_CLUSTER || next_cluster == cluster) {
            break;
        }
        cluster = next_cluster;
    } while (1);
    return formated_clusters;
}

uint8_t _rs_has_invalid_attributes(rs_t *rs) {
    if (rs->sizeof_resource_descriptor_table != (rs->sizeof_resource_descriptor * rs->resource_descriptor_count)) {
        return 1;
    }
    if (rs->sizeof_cluster_table != (rs->sizeof_cluster * rs->cluster_count)) {
        return 2;
    }
    if (rs->sizeof_cluster != (rs->sizeof_cluster_control + rs->sizeof_cluster_data)) {
        return 3;
    }
    if (rs->memory_size != rs->sizeof_cluster_table + rs->cluster_table_address) {
        return 4;
    }
    return 0;
}

```

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

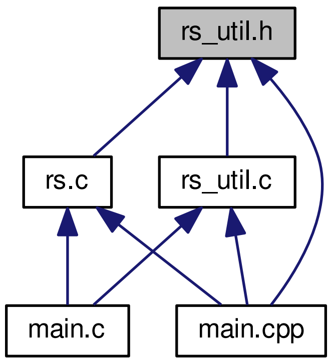
5.29 rs_util.h File Reference

```
#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_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)

5.29.1 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

resource	

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

resource	

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

resource	

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

rs	
resource	

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

rs	
resource	

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

rs	

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

rs	

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

rs	

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

rs	

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

resource	

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

resource	

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

resource	

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

rs	

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

rs	
resource	

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 respecitve flag.

Parameters

resource	

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

rs	
prev_cluster	
next_cluster	

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

rs	
cluster	

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

rs	
cluster	

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

rs	
resource_↩ descriptor	

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

rs	
resource	

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

rs	
cluster	

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

rs	
resource_←descriptor	

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

rs	

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

rs	

Returns

Definition at line 211 of file rs_util.c.

5.29.2.13 uint8_t _rs_is_driver_monted (rs_driver_t *driver*)

Test if given driver is mouted.

Parameters

driver	

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

resource	

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

rs	
cluster	

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

rs	
resource	
offset	

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

rs	
resource	
offset	

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

driver	
rs	

Definition at line 26 of file rs_util.c.

5.29.2.19 void _rs_set_driver_monted (rs_driver_t *driver,* uint8_t *is*)

Set/clear given driver as mouted.

Parameters

driver	
is	

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

driver	
rs	

Definition at line 16 of file rs_util.c.

5.30 rs_util.h

```
#ifndef __SDCC_RS_UTIL_H__
#define __SDCC_RS_UTIL_H__ 1

#include "rs_io.h"
#include "rs.h"

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);

#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)

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);

#define _rs_decrease_free_clusters(rs, n)   { \
                                                rs->free_clusters -= n; \
                                                _rs_write_rs_to_disc(rs->driver, rs); \
                                            }

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

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);

#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,
 CLUSTER_ADDRESS_TO_PREV(address)))

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

uint8_t _rs_has_invalid_attributes(rs_t *rs);

#endif // __SDCC_RS_UTIL_H__
```

5.31 rs_util_spec.h File Reference

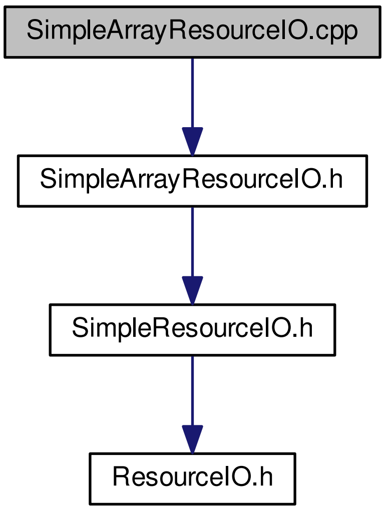
5.32 rs_util_spec.h

```

```

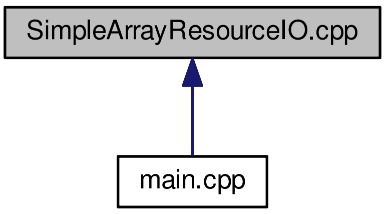
5.33 SimpleArrayResourceIO.cpp File Reference

```
#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.

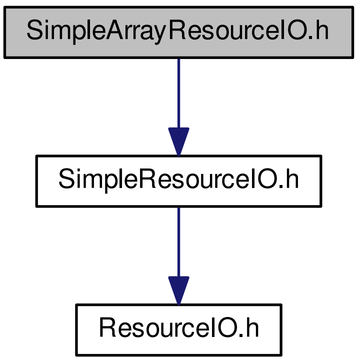
5.34 SimpleArrayResourceIO.cpp

```
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
```

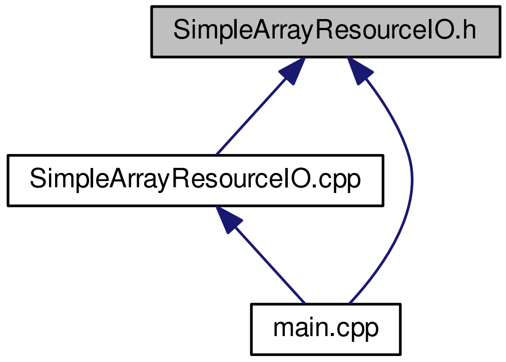
5.35 SimpleArrayResourceIO.h File Reference

`#include <SimpleResourceIO.h>`

Include dependency graph for SimpleArrayResourceIO.h:



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



Classes

- class SimpleArrayResourceIO

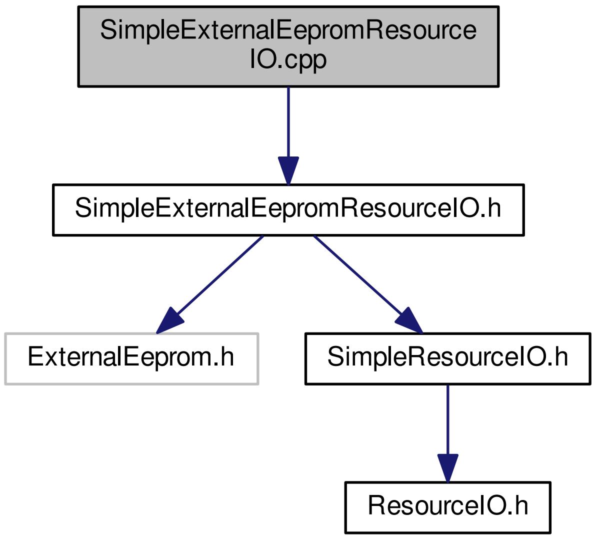
5.36 SimpleArrayResourceIO.h

```
#ifndef __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__
#define __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__ 1

#include <SimpleResourceIO.h>

class SimpleArrayResourceIO : public SimpleResourceIO {
private:
    unsigned char* array;
    unsigned int size;
public:

    SimpleArrayResourceIO(unsigned char* array, unsigned int size);

protected:

    virtual int readBytes(unsigned int address, unsigned char* buf, int len);

    virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
};

#endif  /* __ARDUINO_SIMPLE_ARRAY_RESOURCE_IO_H__ */

```

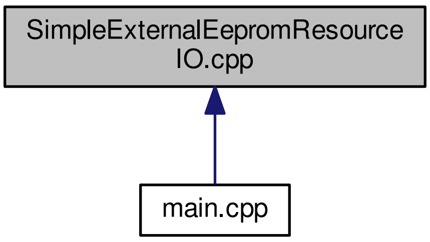
5.37 SimpleExternalEepromResourceIO.cpp File Reference

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

Include dependency graph for SimpleExternalEepromResourceIO.cpp:


SimpleExternalEepromResource IO.cpp
SimpleExternalEepromResourceIO.h
ExternalEeprom.h
SimpleResourceIO.h
ResourceIO.h

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.

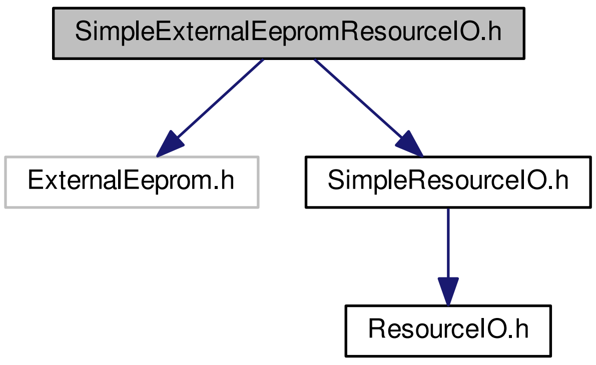
5.38 SimpleExternalEepromResourceIO.cpp

```
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
      (ExternalEeprom* externalEeprom) : SimpleResourceIO(), externalEeprom(externalEeprom) {
00017 }
00018 
00019 int SimpleExternalEepromResourceIO::readBytes(unsigned int address
      , unsigned char* buf, int len) {
00020     return externalEeprom->readBytes(address, buf, len);
00021 }
00022 
00023 void SimpleExternalEepromResourceIO::writeBytes(unsigned int
      address, unsigned char* buf, int len) {
00024     externalEeprom->writeBytes(address, buf, len);
00025 }
00026 
00027 #endif  /* __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_CPP__ */
00028 
```

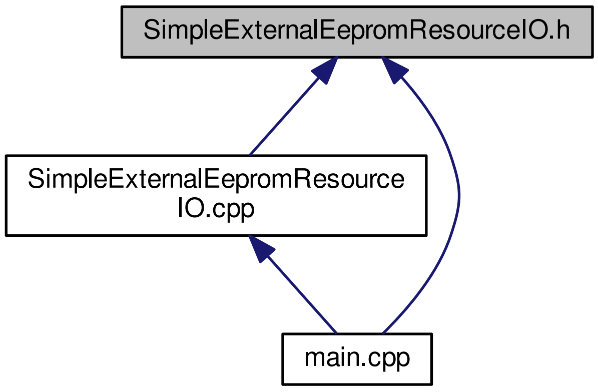
5.39 SimpleExternalEepromResourceIO.h File Reference

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

Include dependency graph for SimpleExternalEepromResourceIO.h:



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


SimpleExternalEepromResourceIO.h
SimpleExternalEepromResource IO.cpp
main.cpp

Classes

- class SimpleExternalEepromResourceIO

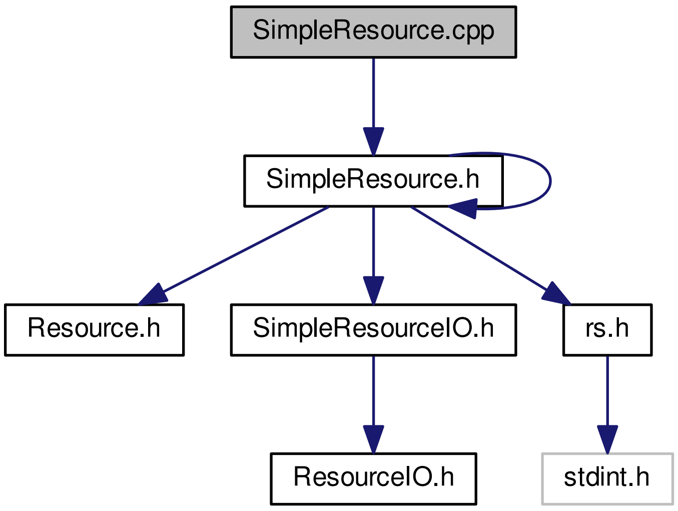
5.40 SimpleExternalEepromResourceIO.h

```
#ifndef __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__
#define __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__ 1

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

class SimpleExternalEepromResourceIO : public
SimpleResourceIO {
private:
    ExternalEeprom* externalEeprom;
public:

    SimpleExternalEepromResourceIO(ExternalEeprom* externalEeprom);

protected:

    virtual int readBytes(unsigned int address, unsigned char* buf, int len);

    virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
};

#endif  /* __ARDUINO_SIMPLE_EXTERNAL_EEPROM_RESOURCE_IO_H__ */

```

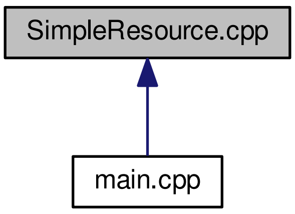
5.41 SimpleResource.cpp File Reference

`#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.

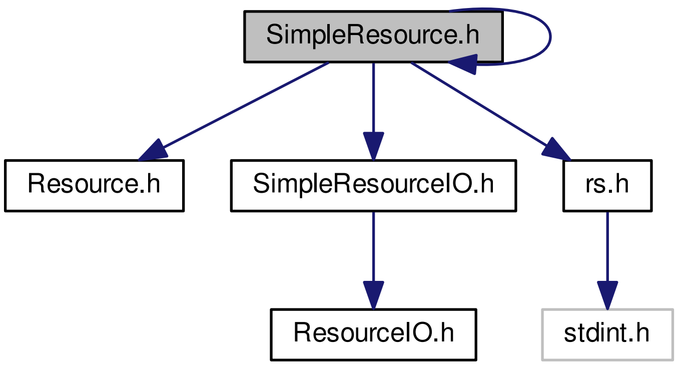
5.42 SimpleResource.cpp

```
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);
    return (lastOperationResult == OPERATION_SUCCESS);
}

void SimpleResource::sync() {
    rs_sync(rs, &resource);
    SimpleResourceIO::getAssociatedIO(rs->
driver)->flush();
}

bool SimpleResource::rewind() {
    lastOperationResult = (ResourceOperationResult)
rs_rewind(rs, &resource);
    return (lastOperationResult == OPERATION_SUCCESS);
}

void SimpleResource::release() {
    sync();
    rs_release(rs, &resource);
}

unsigned int SimpleResource::size() {
    return (unsigned int) rs_size(&resource);
}

unsigned int SimpleResource::tell() {
    return (unsigned int) rs_tell(&resource);
}

bool SimpleResource::eor() {
    return (rs_eor(&resource) != 0);
}

bool SimpleResource::error() {
    return (rs_error(&resource) != 0);
}

bool SimpleResource::isReadOnly() {
    return (rs->flags & RS_RESOURCE_FLAG_BIT_READ_ONLY) != 0;
}

#endif /* __ARDUINO_SIMPLE_RESOURCE_CPP__ */
```

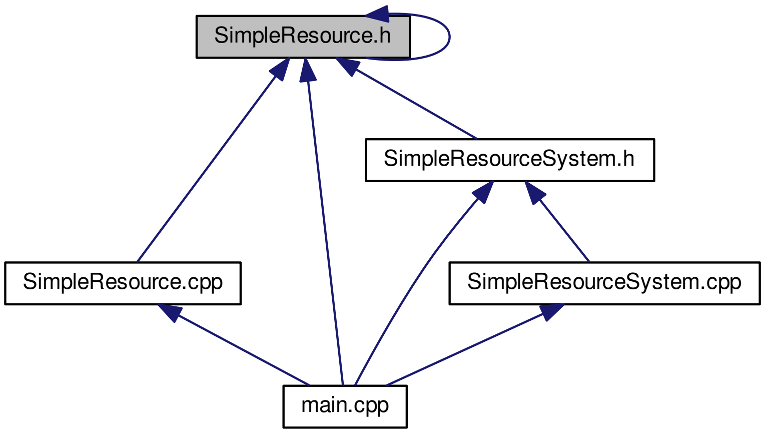
5.43 SimpleResource.h File Reference

```
#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

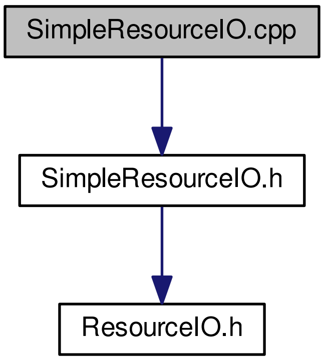
5.44 SimpleResource.h

```
#ifndef __ARDUINO_SIMPLE_RESOURCE_H__
#define __ARDUINO_SIMPLE_RESOURCE_H__ 1

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

class SimpleResource : public Resource {
    rs_resource_code_t code;
    rs_resource_t resource;
    rs_t* rs;
    ResourceOperationResult lastOperationResult;
public:

    SimpleResource(rs_resource_code_t code, rs_t* rs);

    ResourceOperationResult getLastOperationResult() {
        return lastOperationResult;
    }

    virtual void setCode(int code) {
        this->code = (rs_resource_code_t) code;
    }

    virtual int getCode() {
        return (int) this->code;
    }

    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();
};

#endif // __ARDUINO_SIMPLE_RESOURCE_H__
```

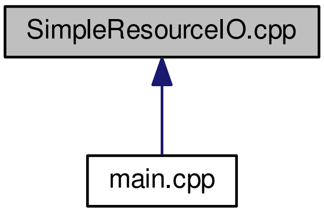
5.45 SimpleResourceIO.cpp File Reference

`#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.

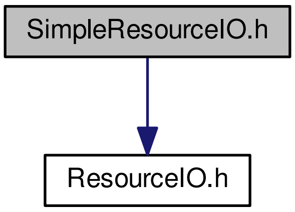
5.46 SimpleResourceIO.cpp

```
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
```

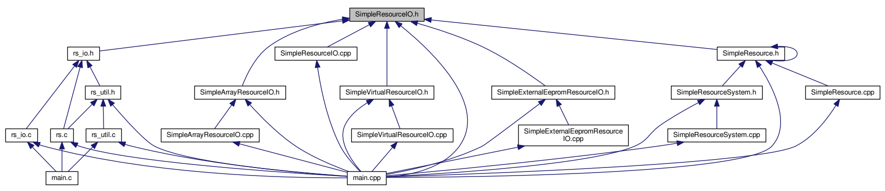
5.47 SimpleResourceIO.h File Reference

`#include <ResourceIO.h>`

Include dependency graph for SimpleResourceIO.h:



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



Classes

- class SimpleResourceIO

Macros

- #define RESOURCE_IO_CACHE_SIZE 8
- #define RESOURCE_IO_DRIVERS_NUM 5

5.47.1 Macro Definition Documentation

5.47.1.1 #define RESOURCE_IO_CACHE_SIZE 8

Arduino - A simple resource implementation.

SimpleResourceIO.h

This is the Resource IO representation.

Author

Dalmir da Silva `dalmirdasilva@gmail.com`

Definition at line 16 of file SimpleResourceIO.h.

5.47.1.2 #define RESOURCE_IO_DRIVERS_NUM 5

Definition at line 17 of file SimpleResourceIO.h.

5.48 SimpleResourceIO.h

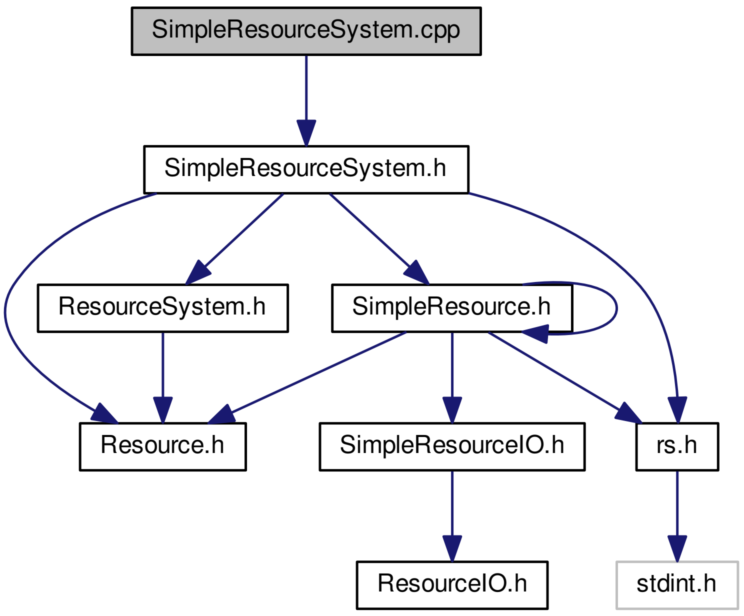
```
00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_IO_H__
00012 #define __ARDUINO_SIMPLE_RESOURCE_IO_H__ 1
00013
00014 #include <ResourceIO.h>
00015
00016 #define RESOURCE_IO_CACHE_SIZE 8
00017 #define RESOURCE_IO_DRIVERS_NUM 5
00018
00019 class SimpleResourceIO : public ResourceIO {
00020 private:
00021
00022     static SimpleResourceIO* association[
      RESOURCE_IO_DRIVERS_NUM];
00023     bool wasCacheChanged, wasCacheInitialized;
00024     unsigned int cacheMemoryAddress;
00025     unsigned char cache[RESOURCE_IO_CACHE_SIZE];
00026     unsigned int cacheMiss, cacheHit;
00027     unsigned int validCacheSize;
00028
00029     void checkCache(unsigned int address) {
00030         if (!wasCacheInitialized || (address < cacheMemoryAddress || address >= (cacheMemoryAddress +
      validCacheSize))) {
00031             flush();
00032             validCacheSize = readBytes(address, cache,
      RESOURCE_IO_CACHE_SIZE);
00033             cacheMemoryAddress = address;
00034             wasCacheChanged = false;
00035             wasCacheInitialized = true;
00036             cacheMiss++;
00037         } else {
00038             cacheHit++;
00039         }
00040     }
00041
00042 protected:
00043
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
```

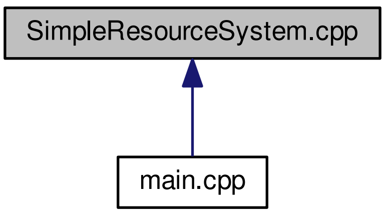
5.49 SimpleResourceSystem.cpp File Reference

`#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

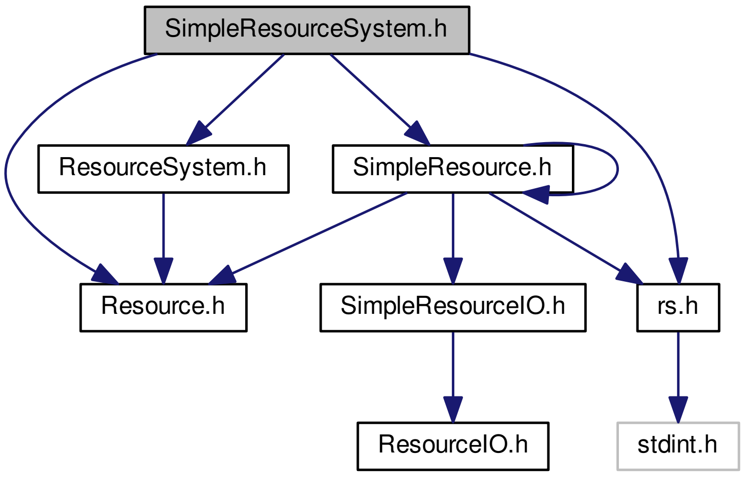
```
00001
00011 #ifndef __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__
00012 #define __ARDUINO_SIMPLE_RESOURCE_SYSTEM_CPP__ 1
00013
00014 #include "SimpleResourceSystem.h"
00015
00016 SimpleResourceSystem::SimpleResourceSystem(int driver) {
00017     lastOperationResult = Resource::OPERATION_SUCCESS;
00018     rs.driver = (rs_driver_t) driver;
00019 }
00020
00021 bool SimpleResourceSystem::mount(MountOptions options) {
00022     lastOperationResult = (Resource::ResourceOperationResult
      ) rs_mount(rs.driver, &rs, (rs_mount_options_t) options);
00023     return (lastOperationResult ==
      Resource::OPERATION_SUCCESS);
00024 }
00025
00026 bool SimpleResourceSystem::umount() {
00027     SimpleResourceIO::getAssociatedIO(rs.
      driver)->flush();
00028     lastOperationResult = (Resource::ResourceOperationResult
      ) rs_umount(&rs);
00029     return (lastOperationResult ==
      Resource::OPERATION_SUCCESS);
00030 }
00031
00032 SimpleResource SimpleResourceSystem::alloc() {
00033     SimpleResource rw(RS_NULL_RESOURCE_CODE, &
      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, &
      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

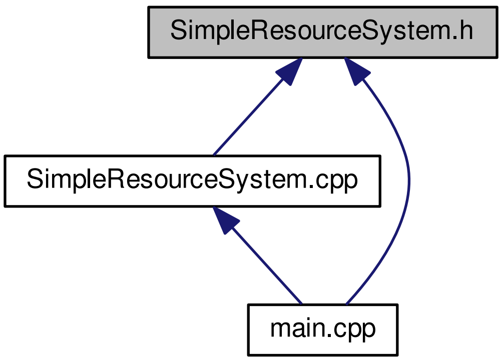
```
#include <Resource.h>
```

```
#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

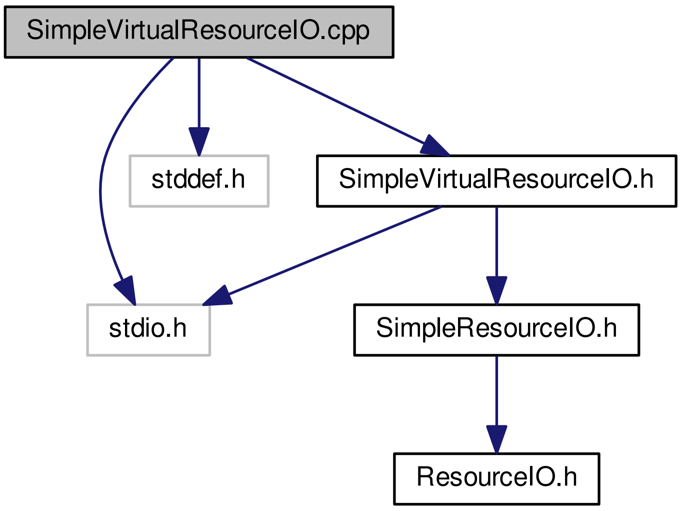
```
00001
```

```
#ifndef __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__
#define __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__ 1

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

class SimpleResourceSystem : public ResourceSystem {
    rs_t rs;
    Resource::ResourceOperationResult
lastOperationResult;
public:

    SimpleResourceSystem(int driver);

    static bool format(rs_t* rs) {
        Resource::ResourceOperationResult o = (
Resource::ResourceOperationResult) rs_format(rs);
        return (o == Resource::OPERATION_SUCCESS);
    }

    rs_t* getRs() {
        return &rs;
    }

    Resource::ResourceOperationResult
getLastOperationResult() {
        return lastOperationResult;
    }

    virtual bool mount(MountOptions options);

    virtual bool umount();

    SimpleResource alloc();

    SimpleResource getResourceByCode(int code);

    virtual unsigned int totalSpace();

    virtual unsigned int availableSpace();
};

#endif // __ARDUINO_SIMPLE_RESOURCE_SYSTEM_H__
```

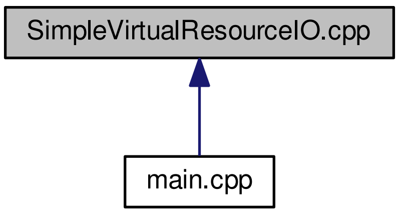
5.53 SimpleVirtualResourceIO.cpp File Reference

```
#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.

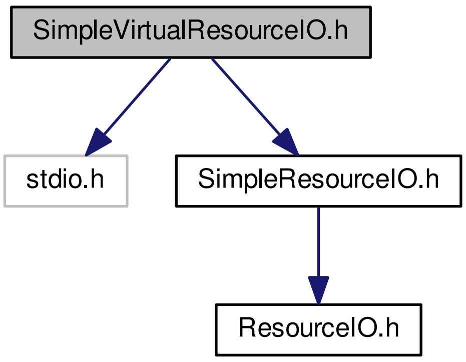
5.54 SimpleVirtualResourceIO.cpp

```
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
```

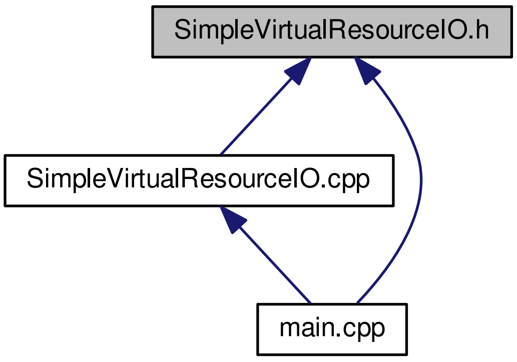
5.55 SimpleVirtualResourceIO.h File Reference

```
#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

```
#ifndef __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__
#define __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__ 1

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

class SimpleVirtualResourceIO : public SimpleResourceIO {
private:
    char *fileName;
    FILE *fp;
public:

    SimpleVirtualResourceIO(char *fileName);

    virtual bool open();

    virtual void flush();

    virtual void close();
protected:

    virtual int readBytes(unsigned int address, unsigned char* buf, int len);

    virtual void writeBytes(unsigned int address, unsigned char* buf, int len);
};

#endif  /* __ARDUINO_SIMPLE_VIRTUAL_RESOURCE_IO_H__ */

```

Index