Arduino Gyroscope Driver

Generated by Doxygen 1.8.9.1

Tue Aug 18 2015 22:52:10

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

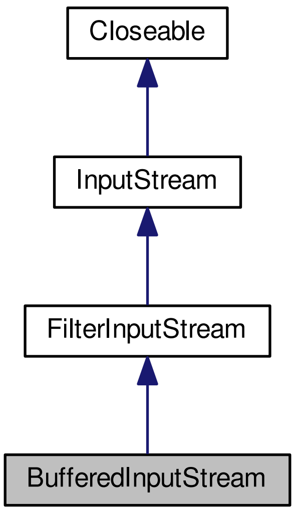
Here is a list of all files with brief descriptions:

4 Class Documentation

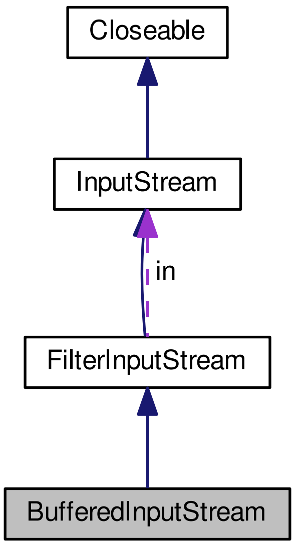
4.1 BufferedInputStream Class Reference

```
#include <BufferedInputStream.h>
```

Inheritance diagram for BufferedInputStream:



Collaboration diagram for BufferedInputStream:



Public Member Functions

- BufferedInputStream (InputStream ∗in, unsigned char ∗buf, int size)
- virtual int available ()
- virtual void close ()
- virtual void mark ()
- virtual bool markSupported ()
- virtual int read ()
- virtual int read (unsigned char ∗b, int len)
- virtual int read (unsigned char ∗b, int off, int len)
- virtual void reset ()
- virtual unsigned int skip (unsigned int n)

Protected Attributes

- unsigned char ∗ buf
- int count
- int pos
- int markpos
- bool marked

Private Member Functions

- void realineBufferContent ()
- void fill (int startPos)

Private Attributes

- unsigned int size

Additional Inherited Members

4.1.1 Detailed Description

Arduino IO.

BufferedInputStream

A `BufferedInputStream` adds functionality to another input stream-namely, the ability to buffer the input and to support the `mark` and `reset` methods. When the `BufferedInputStream` is created, an internal buffer array is passed. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time. The `mark` operation remembers a point in the input stream and the `reset` operation causes all the bytes read since the most recent `mark` operation to be reread before new bytes are taken from the contained input stream.

Definition at line 29 of file BufferedInputStream.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 BufferedInputStream::BufferedInputStream (InputStream ∗ *in*, unsigned char ∗ *buf*, int *size*)

Public constructor.

Parameters

in	
buf	
size	

Definition at line 29 of file BufferedInputStream.cpp.

4.1.3 Member Function Documentation

4.1.3.1 int BufferedInputStream::available ( ) `[virtual]`

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Reimplemented from FilterInputStream.

Definition at line 37 of file BufferedInputStream.cpp.

4.1.3.2 void BufferedInputStream::close ( ) `[virtual]`

Closes this input stream and releases any system resources associated with the stream.

Reimplemented from FilterInputStream.

Definition at line 41 of file BufferedInputStream.cpp.

4.1.3.3 void BufferedInputStream::fill (int *startPos*) `[private]`

Fills the buffer.

Parameters

startPos	

Definition at line 126 of file BufferedInputStream.cpp.

4.1.3.4 void BufferedInputStream::mark ( ) `[virtual]`

Marks the current position in this input stream.

Reimplemented from FilterInputStream.

Definition at line 138 of file BufferedInputStream.cpp.

4.1.3.5 bool BufferedInputStream::markSupported ( ) `[virtual]`

Tests if this input stream supports the mark and reset methods.

Reimplemented from FilterInputStream.

Definition at line 145 of file BufferedInputStream.cpp.

4.1.3.6 int BufferedInputStream::read ( ) `[virtual]`

Reads the next unsigned char of data from the input stream.

Reimplemented from FilterInputStream.

Definition at line 98 of file BufferedInputStream.cpp.

4.1.3.7 int BufferedInputStream::read (unsigned char * *b*, int *len*) `[virtual]`

Reads some number of bytes from the input stream and stores them into the buffer array b.

Parameters

b	
len	

Returns

Reimplemented from FilterInputStream.

Definition at line 51 of file BufferedInputStream.cpp.

4.1.3.8 int BufferedInputStream::read (unsigned char * *b*, int *off*, int *len*) `[virtual]`

Reads some number of bytes from the input stream and stores them into the buffer array b.

Reimplemented from FilterInputStream.

Definition at line 55 of file BufferedInputStream.cpp.

4.1.3.9 void BufferedInputStream::realineBufferContent ( ) `[private]`

Moves the valid bytes on the buffer to the left side of the buffer.

Definition at line 114 of file BufferedInputStream.cpp.

4.1.3.10 void BufferedInputStream::reset ( ) `[virtual]`

Repositions this stream to the position at the time the mark method was last called on this input stream.

Reimplemented from FilterInputStream.

Definition at line 45 of file BufferedInputStream.cpp.

4.1.3.11 unsigned int BufferedInputStream::skip (unsigned int *n*) [virtual]

Skips over and discards n bytes of data from this input stream.

Reimplemented from FilterInputStream.

Definition at line 149 of file BufferedInputStream.cpp.

4.1.4 Member Data Documentation

4.1.4.1 unsigned char* BufferedInputStream::buf [protected]

The internal buffer array where the data is stored.

Definition at line 41 of file BufferedInputStream.h.

4.1.4.2 int BufferedInputStream::count [protected]

The index one greater than the index of the last valid unsigned char in the buffer.

This value is always in the range `0` through `size`; elements `buf[0]` through `buf[count-1]` contain buffered input data obtained from the underlying input stream.

Definition at line 52 of file BufferedInputStream.h.

4.1.4.3 bool BufferedInputStream::marked [protected]

Flag to determine if there is a marker on this input stream.

Definition at line 98 of file BufferedInputStream.h.

4.1.4.4 int BufferedInputStream::markpos [protected]

The value of the `pos` field at the time the last `mark` method was called.

This value is always in the range `0` through `pos`. If there is no marked position in the input stream, this field is `-1`. If there is a marked position in the input stream, then `buf[markpos]` is the first unsigned char to be supplied as input after a `reset` operation. If `markpos` is not `-1`, then all bytes from positions `buf[markpos]` through `buf[pos-1]` must remain in the buffer array (though they may be moved to another place in the buffer array, with suitable adjustments to the values of `count`, `pos`, and `markpos`); they may not be discarded unless and until the difference between `pos` and `markpos` exceeds `marklimit`.

Definition at line 93 of file BufferedInputStream.h.

4.1.4.5 int BufferedInputStream::pos [protected]

The current position in the buffer.

This is the index of the next character to be read from the `buf` array.

This value is always in the range `0` through `count`. If it is less than `count`, then `buf[pos]` is the next unsigned char to be supplied as input; if it is equal to `count`, then the next `read` or `skip` operation will require more bytes to be read from the contained input stream.

Definition at line 67 of file BufferedInputStream.h.

4.1.4.6 unsigned int BufferedInputStream::size [private]

The size of the buffer.

Definition at line 34 of file BufferedInputStream.h.

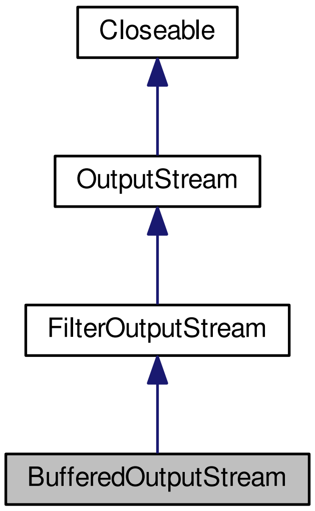
The documentation for this class was generated from the following files:

- BufferedInputStream.h
- BufferedInputStream.cpp

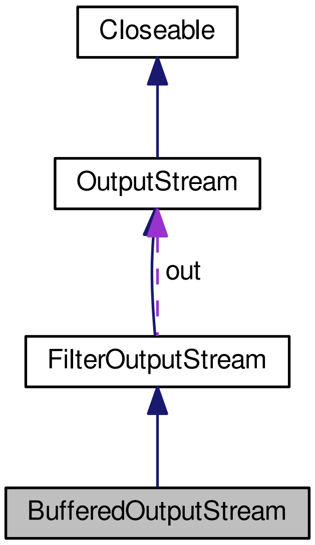
4.2 BufferedOutputStream Class Reference

`#include <BufferedOutputStream.h>`

Inheritance diagram for BufferedOutputStream:



Collaboration diagram for BufferedOutputStream:



Public Member Functions

- BufferedOutputStream (OutputStream ∗out, unsigned char ∗buf, int size)
- void write (unsigned char b)
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char ∗b, int off, int len)
- virtual void flush ()
- virtual void close ()

Protected Attributes

- unsigned char ∗ buf
- int size
- int count

Private Member Functions

- void flushBuffer ()

4.2.1 Detailed Description

Arduino IO.

BufferedOutputStream

The class implements a buffered output stream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each unsigned char written.

Definition at line 17 of file BufferedOutputStream.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BufferedOutputStream::BufferedOutputStream (OutputStream ∗ *out*, unsigned char ∗ *buf*, int *size*)

Creates a new buffered output stream to write data to the specified underlying output stream with the specified buffer size.

Parameters

out	the underlying output stream.
size	the buffer size.

Definition at line 17 of file BufferedOutputStream.cpp.

4.2.3 Member Function Documentation

4.2.3.1 void BufferedOutputStream::close ( ) [virtual]

Closes this output stream and releases any system resources associated with the stream.

The `close` method of `FilterOutputStream` calls its `flush` method, and then calls the `close` method of its underlying output stream.

Reimplemented from FilterOutputStream.

Definition at line 60 of file BufferedOutputStream.cpp.

4.2.3.2 void BufferedOutputStream::flush ( ) [virtual]

Flushes this buffered output stream.

This forces any buffered output bytes to be written out to the underlying output stream.

Reimplemented from FilterOutputStream.

Definition at line 55 of file BufferedOutputStream.cpp.

4.2.3.3 void BufferedOutputStream::flushBuffer ( ) [private]

Flush the internal buffer.

Definition at line 65 of file BufferedOutputStream.cpp.

4.2.3.4 void BufferedOutputStream::write (unsigned char *b*) [virtual]

Writes the specified unsigned char to this buffered output stream.

Parameters

b	the unsigned char to be written.

Exceptions

IOException	if an I/O error occurs.

Reimplemented from FilterOutputStream.

Definition at line 24 of file BufferedOutputStream.cpp.

4.2.3.5 void BufferedOutputStream::write (unsigned char $*$ *b,* int *len*) [virtual]

Writes len bytes from the specified unsigned char array to this output stream.

The general contract for write(b, len) is that it should have exactly the same effect as the call write(b, 0, len).

Parameters

b	
len	

Reimplemented from FilterOutputStream.

Definition at line 31 of file BufferedOutputStream.cpp.

4.2.3.6 void BufferedOutputStream::write (unsigned char $*$ *b,* int *off,* int *len*) [virtual]

Writes `len` bytes from the specified unsigned char array starting at offset `off` to this buffered output stream.

Ordinarily this method stores bytes from the given array into this stream's buffer, flushing the buffer to the underlying output stream as needed. If the requested length is at least as large as this stream's buffer, however, then this method will flush the buffer and write the bytes directly to the underlying output stream. Thus redundant `BufferedOutputStream`s will not copy data unnecessarily.

Parameters

b	the data.
off	the start offset in the data.
len	the number of bytes to write.

Reimplemented from FilterOutputStream.

Definition at line 35 of file BufferedOutputStream.cpp.

4.2.4 Member Data Documentation

4.2.4.1 unsigned char∗ BufferedOutputStream::buf [protected]

The internal buffer where data is stored.

Definition at line 23 of file BufferedOutputStream.h.

4.2.4.2 int BufferedOutputStream::count [protected]

The number of valid bytes in the buffer.

This value is always in the range `0` through `len`; elements `buf[0]` through `buf[count-1]` contain valid unsigned char data.

Definition at line 36 of file BufferedOutputStream.h.

4.2.4.3 int BufferedOutputStream::size [protected]

The size of the buffer where data is stored.

Definition at line 28 of file BufferedOutputStream.h.

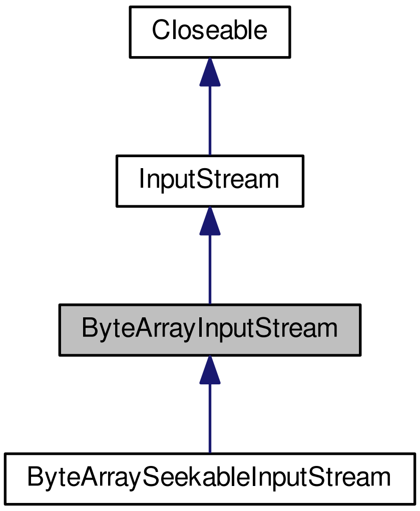
The documentation for this class was generated from the following files:

- BufferedOutputStream.h
- BufferedOutputStream.cpp

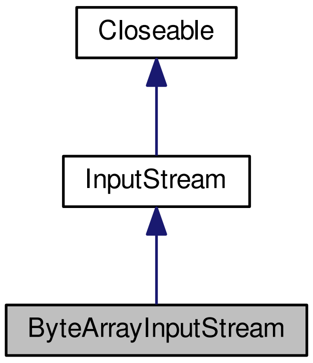
4.3 ByteArrayInputStream Class Reference

`#include <ByteArrayInputStream.h>`

Inheritance diagram for ByteArrayInputStream:



Collaboration diagram for ByteArrayInputStream:



Public Member Functions

- ByteArrayInputStream (unsigned char ∗buf, unsigned int count)
- virtual int available ()
- virtual void mark ()
- virtual bool markSupported ()
- virtual int read ()
- virtual void reset ()

Protected Attributes

- unsigned char ∗ buf
- unsigned int count
- unsigned int pos
- unsigned int markpos

4.3.1 Detailed Description

Arduino IO.

ByteArrayInputStream

A ByteArrayInputStream contains an internal buffer that contains bytes that may be read from the stream.

Definition at line 15 of file ByteArrayInputStream.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ByteArrayInputStream::ByteArrayInputStream (unsigned char ∗ *buf,* unsigned int *count*)

Definition at line 15 of file ByteArrayInputStream.cpp.

4.3.3 Member Function Documentation

4.3.3.1 int ByteArrayInputStream::available ( ) `[virtual]`

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

NOTE: This implementation return 1 or 0. It is because the size of the array is unsigned int, and this method returns a signed int, which means there is no way to return the difference between the current position (can be 0) and the size of the array without possible overflow.

Returns

Reimplemented from InputStream.

Definition at line 22 of file ByteArrayInputStream.cpp.

4.3.3.2 void ByteArrayInputStream::mark ( ) `[virtual]`

Marks the current position in this input stream.

Reimplemented from InputStream.

Definition at line 29 of file ByteArrayInputStream.cpp.

4.3.3.3 bool ByteArrayInputStream::markSupported ( ) `[virtual]`

Tests if this input stream supports the mark and reset methods.

Returns

Reimplemented from InputStream.

Definition at line 33 of file ByteArrayInputStream.cpp.

4.3.3.4 int ByteArrayInputStream::read ( ) `[virtual]`

Reads the next unsigned char of data from the input stream.

Returns

Implements InputStream.

Definition at line 37 of file ByteArrayInputStream.cpp.

4.3.3.5 void ByteArrayInputStream::reset ( ) `[virtual]`

Repositions this stream to the position at the time the mark method was last called on this input stream.

Reimplemented from InputStream.

Definition at line 44 of file ByteArrayInputStream.cpp.

4.3.4 Member Data Documentation

4.3.4.1 unsigned char∗ ByteArrayInputStream::buf `[protected]`

Definition at line 21 of file ByteArrayInputStream.h.

4.3.4.2 unsigned int ByteArrayInputStream::count [protected]

Definition at line 26 of file ByteArrayInputStream.h.

4.3.4.3 unsigned int ByteArrayInputStream::markpos [protected]

Definition at line 36 of file ByteArrayInputStream.h.

4.3.4.4 unsigned int ByteArrayInputStream::pos [protected]

Definition at line 31 of file ByteArrayInputStream.h.

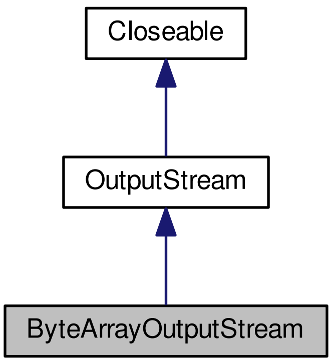
The documentation for this class was generated from the following files:

- ByteArrayInputStream.h
- ByteArrayInputStream.cpp

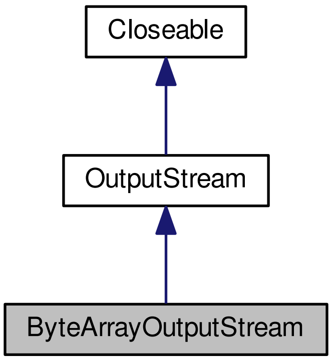
4.4 ByteArrayOutputStream Class Reference

```
#include <ByteArrayOutputStream.h>
```

Inheritance diagram for ByteArrayOutputStream:



Collaboration diagram for ByteArrayOutputStream:



Public Member Functions

- ByteArrayOutputStream (unsigned char ∗buf, unsigned int count)
- void reset ()
- unsigned int size ()
- unsigned char ∗ toByteArray ()
- virtual void write (unsigned char b)

Protected Attributes

- unsigned char ∗ buf
- unsigned int count
- unsigned int pos

4.4.1 Detailed Description

Arduino IO.

ByteArrayOutputStream

This class implements an output stream in which the data is written into a unsigned char array.

Definition at line 15 of file ByteArrayOutputStream.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ByteArrayOutputStream::ByteArrayOutputStream (unsigned char ∗ *buf,* unsigned int *count*)

Public constructor.

Parameters

buf	
count	

Definition at line 15 of file ByteArrayOutputStream.cpp.

4.4.3 Member Function Documentation

4.4.3.1 void ByteArrayOutputStream::reset ( )

Resets the count field of this unsigned char array output stream to zero.

Definition at line 21 of file ByteArrayOutputStream.cpp.

4.4.3.2 unsigned int ByteArrayOutputStream::size ( )

Returns the current size of the buffer.

Returns

unsigned int The size of the stream.

Definition at line 25 of file ByteArrayOutputStream.cpp.

4.4.3.3 unsigned char ∗ ByteArrayOutputStream::toByteArray ( )

Creates a newly allocated unsigned char array.

Returns

unsigned char∗ The unsigned char array.

Definition at line 29 of file ByteArrayOutputStream.cpp.

4.4.3.4 void ByteArrayOutputStream::write (unsigned char *b*) `[virtual]`

Writes the specified unsigned char to this output stream.

Parameters

b	The unsigned char to be written.

Implements OutputStream.

Definition at line 33 of file ByteArrayOutputStream.cpp.

4.4.4 Member Data Documentation

4.4.4.1 unsigned char∗ ByteArrayOutputStream::buf `[protected]`

Definition at line 21 of file ByteArrayOutputStream.h.

4.4.4.2 unsigned int ByteArrayOutputStream::count `[protected]`

Definition at line 26 of file ByteArrayOutputStream.h.

4.4.4.3 unsigned int ByteArrayOutputStream::pos `[protected]`

Definition at line 31 of file ByteArrayOutputStream.h.

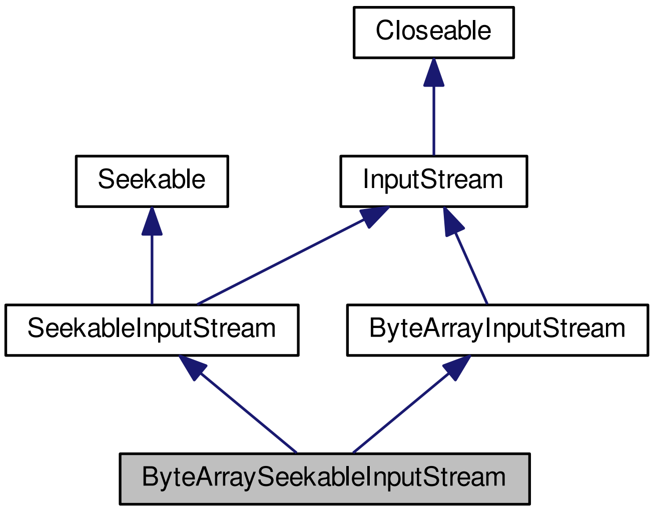
The documentation for this class was generated from the following files:

- ByteArrayOutputStream.h
- ByteArrayOutputStream.cpp

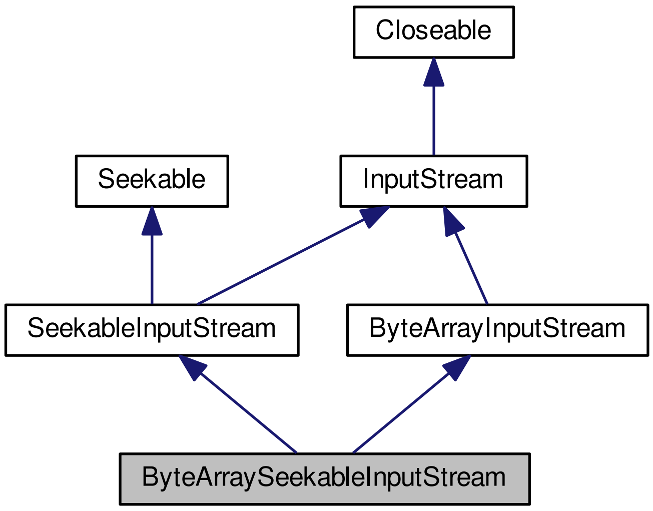
4.5 ByteArraySeekableInputStream Class Reference

```
#include <ByteArraySeekableInputStream.h>
```

Inheritance diagram for ByteArraySeekableInputStream:



Collaboration diagram for ByteArraySeekableInputStream:



Public Member Functions

- ByteArraySeekableInputStream (unsigned char ∗buf, unsigned int count)
- virtual void seek (unsigned int pos)

Additional Inherited Members

4.5.1 Detailed Description

Arduino IO.

ByteArraySeekableInputStream

A ByteArraySeekableInputStream obtains input bytes from a resource in a file system that implements Seekable←↩ InputStream interface.

Definition at line 16 of file ByteArraySeekableInputStream.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ByteArraySeekableInputStream::ByteArraySeekableInputStream (unsigned char ∗ *buf,* unsigned int *count*)

Definition at line 15 of file ByteArraySeekableInputStream.cpp.

4.5.3 Member Function Documentation

4.5.3.1 void ByteArraySeekableInputStream::seek (unsigned int *pos*) `[virtual]`

Implements Seekable.

Definition at line 20 of file ByteArraySeekableInputStream.cpp.

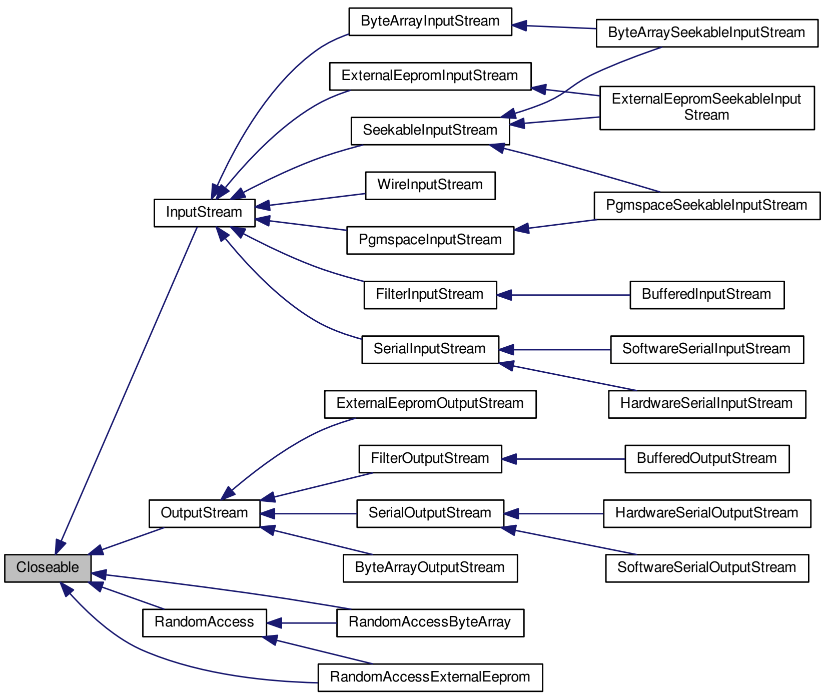
The documentation for this class was generated from the following files:

- ByteArraySeekableInputStream.h
- ByteArraySeekableInputStream.cpp

4.6 Closeable Class Reference

```
#include <Closeable.h>
```

Inheritance diagram for Closeable:


ByteArrayInputStream
ByteArraySeekableInputStream
ExternalEepromInputStream
ExternalEepromSeekableInput Stream
SeekableInputStream
WireInputStream
InputStream
PgmspaceSeekableInputStream
PgmspaceInputStream
FilterInputStream
BufferedInputStream
SerialInputStream
SoftwareSerialInputStream
ExternalEepromOutputStream
HardwareSerialInputStream
FilterOutputStream
BufferedOutputStream
OutputStream
SerialOutputStream
HardwareSerialOutputStream
Closeable
ByteArrayOutputStream
SoftwareSerialOutputStream
RandomAccess
RandomAccessByteArray
RandomAccessExternalEeprom

Public Member Functions

- virtual void close ()=0

4.6.1 Detailed Description

Arduino IO.

Closeable

A Closeable is a source or destination of data that can be closed.

Definition at line 12 of file Closeable.h.

4.6.2 Member Function Documentation

4.6.2.1 virtual void Closeable::close ( ) [pure virtual]

Implemented in BufferedInputStream, FilterInputStream, FilterOutputStream, BufferedOutputStream, Random↩ AccessByteArray, RandomAccessExternalEeprom, InputStream, and OutputStream.

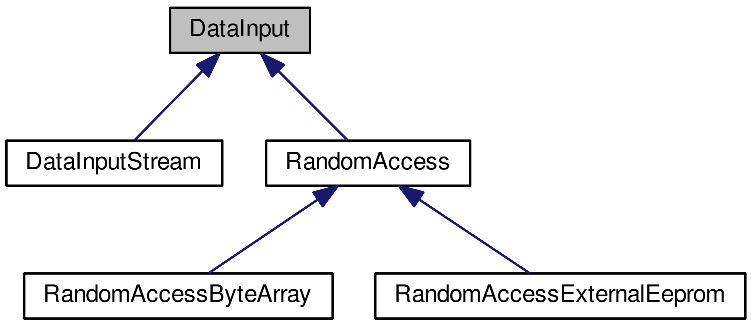
The documentation for this class was generated from the following file:

- Closeable.h

4.7 DataInput Class Reference

```
#include <DataInput.h>
```

Inheritance diagram for DataInput:



Public Member Functions

- virtual unsigned char readByte ()=0
- virtual bool readBoolean ()=0
- virtual char readChar ()=0
- virtual unsigned char readUnsignedChar ()=0
- virtual int readInt ()=0
- virtual unsigned int readUnsignedInt ()=0
- virtual word readWord ()=0
- virtual long readLong ()=0
- virtual unsigned long readUnsignedLong ()=0
- virtual float readFloat ()=0
- virtual double readDouble ()=0
- virtual void readFully (unsigned char ∗b, int len)=0
- virtual unsigned int skipBytes (unsigned int n)=0

4.7.1 Detailed Description

Arduino IO.

DataInput

The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the primitive arduino types.

Definition at line 16 of file DataInput.h.

4.7.2 Member Function Documentation

4.7.2.1 virtual bool DataInput::readBoolean ( ) `[pure virtual]`

Reads a bool from the stream.

Returns

bool

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.2 virtual unsigned char DataInput::readByte ( ) [pure virtual]

Reads a unsigned char from the stream.

Returns

unsigned char

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.3 virtual char DataInput::readChar ( ) [pure virtual]

Reads a char from the stream.

Returns

char

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.4 virtual double DataInput::readDouble ( ) [pure virtual]

Reads a double from the stream.

Returns

double

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.5 virtual float DataInput::readFloat ( ) [pure virtual]

Reads a float from the stream.

Returns

float

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.6 virtual void DataInput::readFully (unsigned char * *b*, int *len*) [pure virtual]

Reads a array of bytes from the stream.

Parameters

b	
len	

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.7 virtual int DataInput::readInt ( ) [pure virtual]

Reads an int from the stream.

Returns

int

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.8 virtual long DataInput::readLong ( ) [pure virtual]

Reads a long from the stream.

Returns

long

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.9 virtual unsigned char DataInput::readUnsignedChar ( ) [pure virtual]

Reads an unsigned char from the stream.

Returns

unsigned char

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.10 virtual unsigned int DataInput::readUnsignedInt ( ) [pure virtual]

Reads an unsigned int from the stream.

Returns

unsigned int

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.11 virtual unsigned long DataInput::readUnsignedLong ( ) [pure virtual]

Reads a unsigned long from the stream.

Returns

unsigned long

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.12 virtual word DataInput::readWord ( ) [pure virtual]

Reads a word from the stream.

Returns

word

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

4.7.2.13 virtual unsigned int DataInput::skipBytes (unsigned int *n*) [pure virtual]

Skips n bytes of the stream.

Parameters

n	

Returns

unsigned int The number of skipped bytes.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataInputStream.

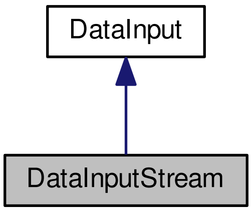
The documentation for this class was generated from the following file:

- DataInput.h

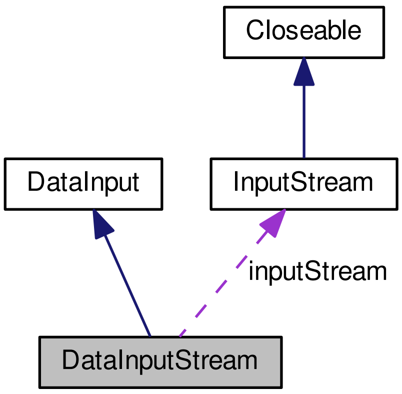
4.8 DataInputStream Class Reference

```
#include <DataInputStream.h>
```

Inheritance diagram for DataInputStream:



Collaboration diagram for DataInputStream:



Public Member Functions

- DataInputStream (InputStream ∗inputStream)
- virtual unsigned char readByte ()
- virtual bool readBoolean ()
- virtual char readChar ()
- virtual unsigned char readUnsignedChar ()
- virtual int readInt ()
- virtual unsigned int readUnsignedInt ()
- virtual word readWord ()
- virtual long readLong ()
- virtual unsigned long readUnsignedLong ()
- virtual float readFloat ()

- virtual double readDouble ()
- virtual void readFully (unsigned char ∗b, int len)
- virtual unsigned int skipBytes (unsigned int n)

Private Attributes

- InputStream ∗ inputStream

4.8.1 Detailed Description

Arduino IO.

DataInputStream

A data input stream lets an application read data from a InputStream.

Definition at line 15 of file DataInputStream.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 DataInputStream::DataInputStream (InputStream ∗ *inputStream*)

Public constructor.

Parameters

inputStream	

Definition at line 14 of file DataInputStream.cpp.

4.8.3 Member Function Documentation

4.8.3.1 bool DataInputStream::readBoolean ( ) [virtual]

Reads a bool from the stream.

Returns

bool

Implements DataInput.

Definition at line 22 of file DataInputStream.cpp.

4.8.3.2 unsigned char DataInputStream::readByte ( ) [virtual]

Reads a unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 18 of file DataInputStream.cpp.

4.8.3.3 char DataInputStream::readChar ( ) [virtual]

Reads a char from the stream.

Returns

char

Implements DataInput.

Definition at line 26 of file DataInputStream.cpp.

4.8.3.4 double DataInputStream::readDouble ( ) `[virtual]`

Reads a double from the stream.

Returns

double

Implements DataInput.

Definition at line 70 of file DataInputStream.cpp.

4.8.3.5 float DataInputStream::readFloat ( ) `[virtual]`

Reads a float from the stream.

Returns

float

Implements DataInput.

Definition at line 66 of file DataInputStream.cpp.

4.8.3.6 void DataInputStream::readFully (unsigned char ∗ *b*, int *len*) `[virtual]`

Reads a array of bytes from the stream.

Parameters

b	
len	

Implements DataInput.

Definition at line 74 of file DataInputStream.cpp.

4.8.3.7 int DataInputStream::readInt ( ) `[virtual]`

Reads an int from the stream.

Returns

int

Implements DataInput.

Definition at line 34 of file DataInputStream.cpp.

4.8.3.8 long DataInputStream::readLong ( ) `[virtual]`

Reads a long from the stream.

Returns

long

Implements DataInput.

Definition at line 50 of file DataInputStream.cpp.

4.8.3.9 unsigned char DataInputStream::readUnsignedChar ( ) [virtual]

Reads an unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 30 of file DataInputStream.cpp.

4.8.3.10 unsigned int DataInputStream::readUnsignedInt ( ) [virtual]

Reads an unsigned int from the stream.

Returns

unsigned int

Implements DataInput.

Definition at line 42 of file DataInputStream.cpp.

4.8.3.11 unsigned long DataInputStream::readUnsignedLong ( ) [virtual]

Reads a unsigned long from the stream.

Returns

unsigned long

Implements DataInput.

Definition at line 62 of file DataInputStream.cpp.

4.8.3.12 word DataInputStream::readWord ( ) [virtual]

Reads a word from the stream.

Returns

word

Implements DataInput.

Definition at line 46 of file DataInputStream.cpp.

4.8.3.13 unsigned int DataInputStream::skipBytes (unsigned int *n*) [virtual]

Skips n bytes of the stream.

Parameters

n	

Returns

unsigned int The number of skipped bytes.

Implements DataInput.

Definition at line 80 of file DataInputStream.cpp.

4.8.4 Member Data Documentation

4.8.4.1 InputStream∗ DataInputStream::inputStream [private]

The used input stream.

Definition at line 20 of file DataInputStream.h.

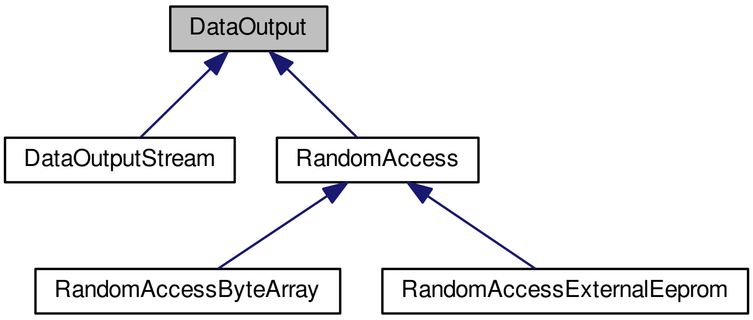
The documentation for this class was generated from the following files:

- DataInputStream.h
- DataInputStream.cpp

4.9 DataOutput Class Reference

```
#include <DataOutput.h>
```

Inheritance diagram for DataOutput:



Public Member Functions

- virtual void write (unsigned char ∗b, int len)=0
- virtual void write (unsigned char b)=0
- virtual void writeByte (unsigned char b)=0
- virtual void writeBytes (unsigned char ∗b, int len)=0
- virtual void writeBoolean (bool v)=0
- virtual void writeChar (char c)=0
- virtual void writeUnsignedChar (unsigned char c)=0
- virtual void writeInt (int v)=0
- virtual void writeUnsignedInt (unsigned int v)=0
- virtual void writeWord (word v)=0
- virtual void writeLong (long v)=0
- virtual void writeUnsignedLong (unsigned long v)=0
- virtual void writeFloat (float v)=0
- virtual void writeDouble (double v)=0

4.9.1 Detailed Description

Arduino IO.

DataOutput

The DataOutput interface provides for converting data from any of the primitive types to a series of bytes and writing these bytes to a binary stream.

Definition at line 15 of file DataOutput.h.

4.9.2 Member Function Documentation

4.9.2.1 virtual void DataOutput::write (unsigned char $*$ *b*, int *len*) `[pure virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.2 virtual void DataOutput::write (unsigned char *b*) `[pure virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.3 virtual void DataOutput::writeBoolean (bool *v*) `[pure virtual]`

Writes a bool into the stream.

Parameters

v	The bool to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.4 virtual void DataOutput::writeByte (unsigned char *b*) `[pure virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.5 virtual void DataOutput::writeBytes (unsigned char $*$ *b*, int *len*) `[pure virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.6 virtual void DataOutput::writeChar (char *c*) `[pure virtual]`

Writes a char into the stream.

Parameters

c	The char to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.7 virtual void DataOutput::writeDouble (double *v*) `[pure virtual]`

Writes a double into the stream.

Parameters

v	The double to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.8 virtual void DataOutput::writeFloat (float *v*) `[pure virtual]`

Writes a float into the stream.

Parameters

v	The float to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.9 virtual void DataOutput::writeInt (int *v*) `[pure virtual]`

Writes an int into the stream.

Parameters

v	The int to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.10 virtual void DataOutput::writeLong (long *v*) `[pure virtual]`

Writes a long into the stream.

Parameters

v	The long to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.11 virtual void DataOutput::writeUnsignedChar (unsigned char *c*) `[pure virtual]`

Writes an unsigned char into the stream.

Parameters

c	The unsigned char to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.12 virtual void DataOutput::writeUnsignedInt (unsigned int *v*) `[pure virtual]`

Writes an unsigned int into the stream.

Parameters

v	The unsigned int to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.13 virtual void DataOutput::writeUnsignedLong (unsigned long *v*) `[pure virtual]`

Writes a unsigned long into the stream.

Parameters

v	The unsigned long to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

4.9.2.14 virtual void DataOutput::writeWord (word *v*) `[pure virtual]`

Writes a word into the stream.

Parameters

v	The word to be written.

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, and DataOutputStream.

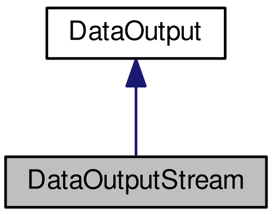
The documentation for this class was generated from the following file:

- DataOutput.h

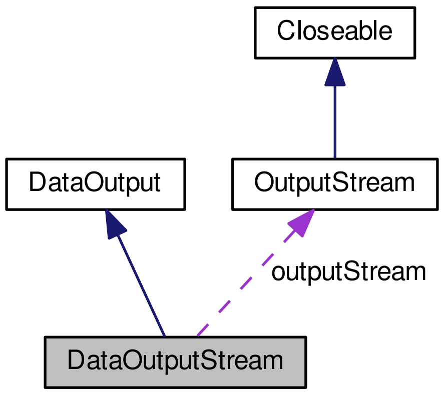
4.10 DataOutputStream Class Reference

```
#include <DataOutputStream.h>
```

Inheritance diagram for DataOutputStream:



Collaboration diagram for DataOutputStream:



Public Member Functions

- DataOutputStream (OutputStream ∗outputStream)
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char b)
- virtual void writeByte (unsigned char b)
- virtual void writeBytes (unsigned char ∗b, int len)
- virtual void writeBoolean (bool v)
- virtual void writeChar (char c)
- virtual void writeUnsignedChar (unsigned char c)
- virtual void writeInt (int v)
- virtual void writeUnsignedInt (unsigned int v)
- virtual void writeWord (word v)
- virtual void writeLong (long v)
- virtual void writeUnsignedLong (unsigned long v)
- virtual void writeFloat (float v)
- virtual void writeDouble (double v)

Private Attributes

- OutputStream ∗ outputStream

4.10.1 Detailed Description

Arduino IO.

DataOutputStream

A data output stream lets an application write types to an OutputStream.

Definition at line 16 of file DataOutputStream.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 DataOutputStream::DataOutputStream (**OutputStream** $*$ *outputStream*)

Public constructor.

Parameters

outputStream	The stream to be used.

Definition at line 14 of file DataOutputStream.cpp.

4.10.3 Member Function Documentation

4.10.3.1 void DataOutputStream::write (unsigned char ∗ *b*, int *len*) `[virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 18 of file DataOutputStream.cpp.

4.10.3.2 void DataOutputStream::write (unsigned char *b*) `[virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 22 of file DataOutputStream.cpp.

4.10.3.3 void DataOutputStream::writeBoolean (bool *v*) `[virtual]`

Writes a bool into the stream.

Parameters

v	The bool to be written.

Implements DataOutput.

Definition at line 36 of file DataOutputStream.cpp.

4.10.3.4 void DataOutputStream::writeByte (unsigned char *b*) `[virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 26 of file DataOutputStream.cpp.

4.10.3.5 void DataOutputStream::writeBytes (unsigned char ∗ *b*, int *len*) `[virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 30 of file DataOutputStream.cpp.

4.10.3.6 void DataOutputStream::writeChar (char *c*) `[virtual]`

Writes a char into the stream.

Parameters

c	The char to be written.

Implements DataOutput.

Definition at line 40 of file DataOutputStream.cpp.

4.10.3.7 void DataOutputStream::writeDouble (double *v*) `[virtual]`

Writes a double into the stream.

Parameters

v	The double to be written.

Implements DataOutput.

Definition at line 76 of file DataOutputStream.cpp.

4.10.3.8 void DataOutputStream::writeFloat (float *v*) `[virtual]`

Writes a float into the stream.

Parameters

v	The float to be written.

Implements DataOutput.

Definition at line 72 of file DataOutputStream.cpp.

4.10.3.9 void DataOutputStream::writeInt (int *v*) `[virtual]`

Writes an int into the stream.

Parameters

v	The int to be written.

Implements DataOutput.

Definition at line 48 of file DataOutputStream.cpp.

4.10.3.10 void DataOutputStream::writeLong (long *v*) `[virtual]`

Writes a long into the stream.

Parameters

v	The long to be written.

Implements DataOutput.

Definition at line 61 of file DataOutputStream.cpp.

4.10.3.11 void DataOutputStream::writeUnsignedChar (unsigned char *c*) `[virtual]`

Writes an unsigned char into the stream.

Parameters

c	The unsigned char to be written.

Implements DataOutput.

Definition at line 44 of file DataOutputStream.cpp.

4.10.3.12 void DataOutputStream::writeUnsignedInt (unsigned int *v*) [virtual]

Writes an unsigned int into the stream.

Parameters

v	The unsigned int to be written.

Implements DataOutput.

Definition at line 53 of file DataOutputStream.cpp.

4.10.3.13 void DataOutputStream::writeUnsignedLong (unsigned long *v*) [virtual]

Writes a unsigned long into the stream.

Parameters

v	The unsigned long to be written.

Implements DataOutput.

Definition at line 68 of file DataOutputStream.cpp.

4.10.3.14 void DataOutputStream::writeWord (word *v*) [virtual]

Writes a word into the stream.

Parameters

v	The word to be written.

Implements DataOutput.

Definition at line 57 of file DataOutputStream.cpp.

4.10.4 Member Data Documentation

4.10.4.1 OutputStream* DataOutputStream::outputStream [private]

The stream to be used.

Definition at line 21 of file DataOutputStream.h.

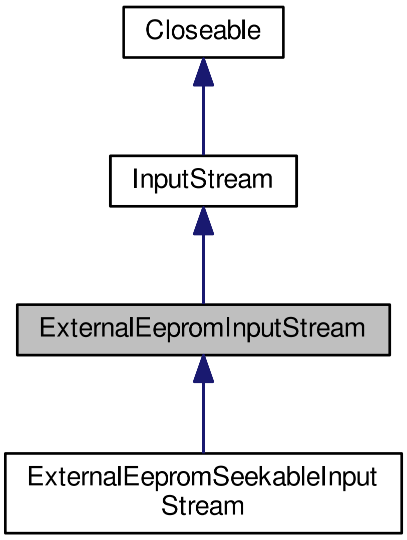
The documentation for this class was generated from the following files:

- DataOutputStream.h
- DataOutputStream.cpp

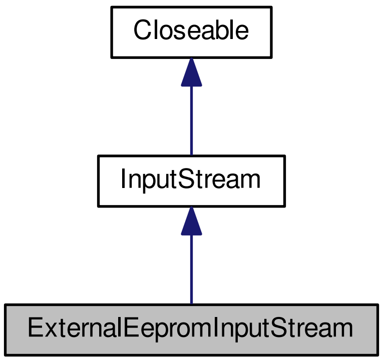
4.11 ExternalEepromInputStream Class Reference

```
#include <ExternalEepromInputStream.h>
```

Inheritance diagram for ExternalEepromInputStream:



Collaboration diagram for ExternalEepromInputStream:



Public Member Functions

- ExternalEepromInputStream (ExternalEeprom ∗externalEeprom)
- virtual int available ()
- virtual void mark ()
- virtual bool markSupported ()
- virtual int read ()
- virtual int read (unsigned char ∗b, int off, int len)
- virtual void reset ()

Protected Attributes

- ExternalEeprom ∗ externalEeprom
- unsigned int pos
- unsigned int markpos
- unsigned int externalEepromSize

4.11.1 Detailed Description

Arduino IO.

ExternalEepromInputStream

An ExternalEepromInputStream obtains input bytes from a externalEeprom.

Definition at line 16 of file ExternalEepromInputStream.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ExternalEepromInputStream::ExternalEepromInputStream (ExternalEeprom ∗ *externalEeprom*)

Public constructor.

Parameters

externalEeprom	The externalEeprom where data is stored.

Definition at line 15 of file ExternalEepromInputStream.cpp.

4.11.3 Member Function Documentation

4.11.3.1 int ExternalEepromInputStream::available ( ) [virtual]

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Returns

int The available number of bytes.

Reimplemented from InputStream.

Definition at line 23 of file ExternalEepromInputStream.cpp.

4.11.3.2 void ExternalEepromInputStream::mark ( ) [virtual]

Marks the current position in this input stream.

Reimplemented from InputStream.

Definition at line 30 of file ExternalEepromInputStream.cpp.

4.11.3.3 bool ExternalEepromInputStream::markSupported ( ) [virtual]

Tests if this input stream supports the mark and reset methods.

Returns

bool

Reimplemented from InputStream.

Definition at line 34 of file ExternalEepromInputStream.cpp.

4.11.3.4 int ExternalEepromInputStream::read ( ) [virtual]

Reads the next unsigned char of data from the input stream.

Returns

int The read unsigned char as an int.

Implements InputStream.

Definition at line 38 of file ExternalEepromInputStream.cpp.

4.11.3.5 int ExternalEepromInputStream::read (unsigned char ∗ *b*, int *off*, int *len*) [virtual]

Reads len of bytes from the input stream.

Parameters

b	
off	
len	

Returns

Reimplemented from InputStream.

Definition at line 45 of file ExternalEepromInputStream.cpp.

4.11.3.6 void ExternalEepromInputStream::reset ( ) [virtual]

Repositions this stream to the position at the time the mark method was last called on this input stream.

Reimplemented from InputStream.

Definition at line 54 of file ExternalEepromInputStream.cpp.

4.11.4 Member Data Documentation

4.11.4.1 ExternalEeprom∗ ExternalEepromInputStream::externalEeprom [protected]

Definition at line 22 of file ExternalEepromInputStream.h.

4.11.4.2 unsigned int ExternalEepromInputStream::externalEepromSize [protected]

Definition at line 37 of file ExternalEepromInputStream.h.

4.11.4.3 unsigned int ExternalEepromInputStream::markpos [protected]

Definition at line 32 of file ExternalEepromInputStream.h.

4.11.4.4 unsigned int ExternalEepromInputStream::pos [protected]

Definition at line 27 of file ExternalEepromInputStream.h.

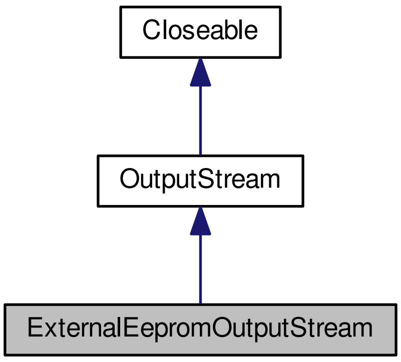
The documentation for this class was generated from the following files:

- ExternalEepromInputStream.h
- ExternalEepromInputStream.cpp

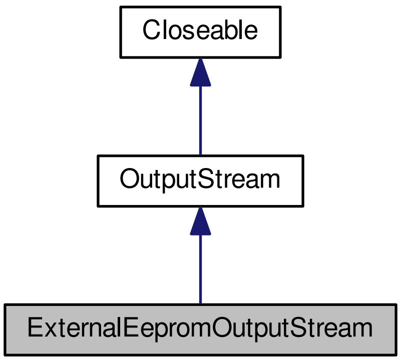
4.12 ExternalEepromOutputStream Class Reference

`#include <ExternalEepromOutputStream.h>`

Inheritance diagram for ExternalEepromOutputStream:



Collaboration diagram for ExternalEepromOutputStream:



Public Member Functions

- ExternalEepromOutputStream (ExternalEeprom ∗externalEeprom)
- virtual void write (unsigned char b)
- virtual void write (unsigned char ∗b, int off, int len)

Private Attributes

- ExternalEeprom ∗ externalEeprom
- unsigned int pos

4.12.1 Detailed Description

Arduino IO.

ExternalEepromOutputStream

A resource output stream is an output stream for writing data to an ExternalEeprom.

Definition at line 16 of file ExternalEepromOutputStream.h.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 ExternalEepromOutputStream::ExternalEepromOutputStream (ExternalEeprom $*$ *externalEeprom*)

Public constructor.

Parameters

externalEeprom	

Definition at line 14 of file ExternalEepromOutputStream.cpp.

4.12.3 Member Function Documentation

4.12.3.1 void ExternalEepromOutputStream::write (unsigned char *b*) `[virtual]`

Writes the specified unsigned char to this output stream.

Parameters

b	

Implements OutputStream.

Definition at line 20 of file ExternalEepromOutputStream.cpp.

4.12.3.2 void ExternalEepromOutputStream::write (unsigned char $*$ *b*, int *off*, int *len*) `[virtual]`

Writes len bytes from the specified unsigned char array starting at offset off to this output stream.

Parameters

b	
off	
len	

Reimplemented from OutputStream.

Definition at line 24 of file ExternalEepromOutputStream.cpp.

4.12.4 Member Data Documentation

4.12.4.1 ExternalEeprom$*$ ExternalEepromOutputStream::externalEeprom `[private]`

The associated eeprom.

Definition at line 21 of file ExternalEepromOutputStream.h.

4.12.4.2 unsigned int ExternalEepromOutputStream::pos `[private]`

Current eeprom position.

Definition at line 26 of file ExternalEepromOutputStream.h.

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

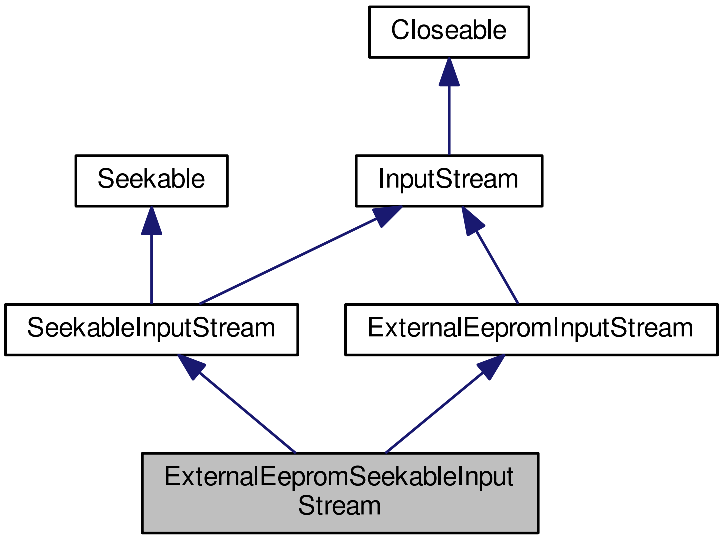
- ExternalEepromOutputStream.h

- ExternalEepromOutputStream.cpp

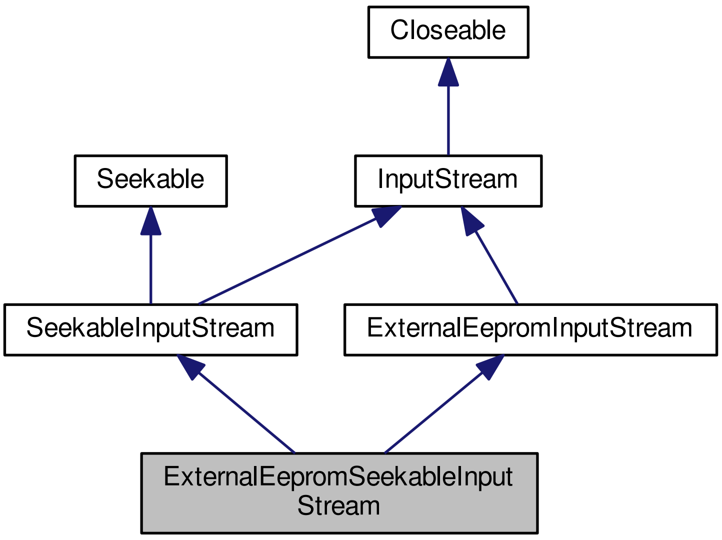
4.13 ExternalEepromSeekableInputStream Class Reference

```
#include <ExternalEepromSeekableInputStream.h>
```

Inheritance diagram for ExternalEepromSeekableInputStream:



Collaboration diagram for ExternalEepromSeekableInputStream:



Public Member Functions

- ExternalEepromSeekableInputStream (ExternalEeprom ∗externalEeprom)
- virtual void seek (unsigned int pos)

Additional Inherited Members

4.13.1 Detailed Description

Arduino IO.

ExternalEepromSeekableInputStream

A ExternalEepromSeekableInputStream obtains input bytes from a external input stream.

Definition at line 17 of file ExternalEepromSeekableInputStream.h.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 ExternalEepromSeekableInputStream::ExternalEepromSeekableInputStream (ExternalEeprom ∗ *externalEeprom*)

Public constructor.

Parameters

resource	The external eeprom to be used.

Definition at line 15 of file ExternalEepromSeekableInputStream.cpp.

4.13.3 Member Function Documentation

4.13.3.1 void ExternalEepromSeekableInputStream::seek (unsigned int *pos*) `[virtual]`

Seeks this input stream to the position.

Parameters

pos	THe position.

Implements Seekable.

Definition at line 20 of file ExternalEepromSeekableInputStream.cpp.

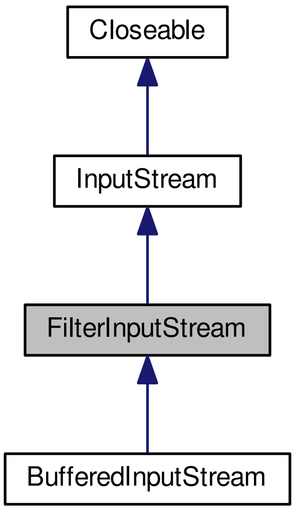
The documentation for this class was generated from the following files:

- ExternalEepromSeekableInputStream.h
- ExternalEepromSeekableInputStream.cpp

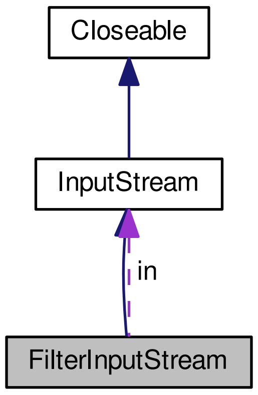
4.14 FilterInputStream Class Reference

```
#include <FilterInputStream.h>
```

Inheritance diagram for FilterInputStream:



Collaboration diagram for FilterInputStream:



Public Member Functions

- virtual int read ()
- virtual int read (unsigned char ∗b, int len)
- virtual int read (unsigned char ∗b, int off, int len)
- virtual unsigned int skip (unsigned int n)
- virtual int available ()
- virtual void close ()
- virtual void mark ()
- virtual void reset ()
- virtual bool markSupported ()

Protected Member Functions

- FilterInputStream (InputStream ∗in)

Protected Attributes

- InputStream ∗ in

4.14.1 Detailed Description

A `FilterInputStream` contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.

The class `FilterInputStream` itself simply overrides all methods of `InputStream` with versions that pass all requests to the contained input stream. Subclasses of `FilterInputStream` may further override some of these methods and may also provide additional methods and fields.

Definition at line 21 of file FilterInputStream.h.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 FilterInputStream::FilterInputStream (**InputStream** ∗ *in*) `[protected]`

Creates a `FilterInputStream` by assigning the argument `in` to the field `this->in` so as to remember it for later use.

Parameters

in	the underlying input stream

Definition at line 21 of file FilterInputStream.cpp.

4.14.3 Member Function Documentation

4.14.3.1 int FilterInputStream::available ( ) [virtual]

Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Returns

an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 41 of file FilterInputStream.cpp.

4.14.3.2 void FilterInputStream::close ( ) [virtual]

Closes this input stream.

This method simply performs `in->close()`.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 45 of file FilterInputStream.cpp.

4.14.3.3 void FilterInputStream::mark ( ) [virtual]

Marks the current position in this input stream.

A subsequent call to the `reset` method repositions this stream at the last marked position so that subsequent reads re-read the same bytes.

This method simply performs `in->mark()`.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 49 of file FilterInputStream.cpp.

4.14.3.4 bool FilterInputStream::markSupported ( ) [virtual]

Tests if this input stream supports the `mark` and `reset` methods.

This method simply performs `in->markSupported()`.

Returns

`true` if this stream type supports the `mark` and `reset` method; `false` otherwise.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 57 of file FilterInputStream.cpp.

4.14.3.5 int FilterInputStream::read ( ) [virtual]

Reads the next unsigned char of data from this input stream.

The value unsigned char is returned as an `int` in the range `0` to `255`. If no unsigned char is available because the end of the stream has been reached, the value `-1` is returned.

This method simply performs `in->read()` and returns the result.

Returns

the next unsigned char of data, or `-1` if the end of the stream is reached.

Implements InputStream.

Reimplemented in BufferedInputStream.

Definition at line 25 of file FilterInputStream.cpp.

4.14.3.6 int FilterInputStream::read (unsigned char ∗ *b*, int *len*) [virtual]

Reads up to `len` bytes of data from this input stream into an array of bytes.

This method simply performs the call `read(b, 0, len)` and returns the result. It is important that it does *not* do `in->read(b)` instead; certain subclasses of `FilterInputStream` depend on the implementation strategy actually used.

Parameters

b	the buffer into which the data is read.

Returns

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 29 of file FilterInputStream.cpp.

4.14.3.7 int FilterInputStream::read (unsigned char ∗ *b*, int *off*, int *len*) [virtual]

Reads up to `len` bytes of data from this input stream into an array of bytes.

This method simply performs `in->read(b, off, len)` and returns the result.

Parameters

b	the buffer into which the data is read.
off	the start offset in the destination array `b`
len	the maximum number of bytes read.

Returns

the total number of bytes read into the buffer, or `-1` if there is no more data because the end of the stream has been reached.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 33 of file FilterInputStream.cpp.

4.14.3.8 void FilterInputStream::reset ( ) [virtual]

Repositions this stream to the position at the time the `mark` method was last called on this input stream.

This method simply performs `in->reset()`.

Stream marks are intended to be used in situations where you need to read ahead a little to see what's in the stream. Often this is most easily done by invoking some general parser. If the stream is of the type handled by the parse, it just chugs along happily. If the stream is not of that type, the parser should toss an exception when it fails.

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 53 of file FilterInputStream.cpp.

4.14.3.9 unsigned int FilterInputStream::skip (unsigned int *n*) [virtual]

This method simply performs `in->skip(n)`.

Parameters

Reimplemented from InputStream.

Reimplemented in BufferedInputStream.

Definition at line 37 of file FilterInputStream.cpp.

4.14.4 Member Data Documentation

4.14.4.1 InputStream* FilterInputStream::in [protected]

The input stream to be filtered.

Definition at line 28 of file FilterInputStream.h.

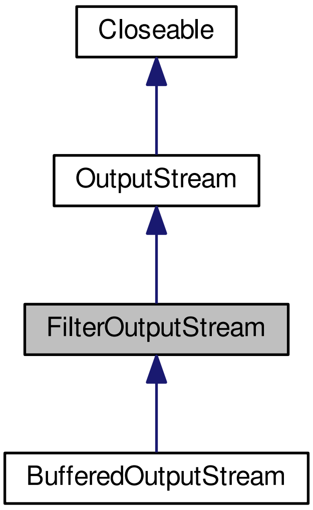
The documentation for this class was generated from the following files:

- FilterInputStream.h
- FilterInputStream.cpp

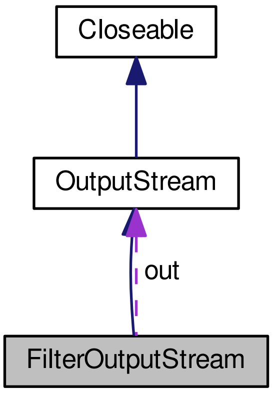
4.15 FilterOutputStream Class Reference

`#include <FilterOutputStream.h>`

Inheritance diagram for FilterOutputStream:



Collaboration diagram for FilterOutputStream:



Public Member Functions

- FilterOutputStream (OutputStream ∗out)
- virtual void write (unsigned char b)
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char ∗b, int off, int len)
- virtual void flush ()
- virtual void close ()

Protected Attributes

- OutputStream $*$ out

4.15.1 Detailed Description

Arduino IO.

FilterOutputStream

This class is the superclass of all classes that filter output streams. These streams sit on top of an already existing output stream (the *underlying* output stream) which it uses as its basic sink of data, but possibly transforming the data along the way or providing additional functionality.

The class `FilterOutputStream` itself simply overrides all methods of `OutputStream` with versions that pass all requests to the underlying output stream. Subclasses of `FilterOutputStream` may further override some of these methods as well as provide additional methods and fields.

Definition at line 24 of file FilterOutputStream.h.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 FilterOutputStream::FilterOutputStream (**OutputStream** $*$ *out*)

Creates an output stream filter built on top of the specified underlying output stream.

Parameters

out	the underlying output stream to be assigned to the field `this->out` for later use.

Definition at line 24 of file FilterOutputStream.cpp.

4.15.3 Member Function Documentation

4.15.3.1 void FilterOutputStream::close ( ) `[virtual]`

Closes this output stream and releases any system resources associated with the stream.

The `close` method of `FilterOutputStream` calls its `flush` method, and then calls the `close` method of its underlying output stream.

Reimplemented from OutputStream.

Reimplemented in BufferedOutputStream.

Definition at line 44 of file FilterOutputStream.cpp.

4.15.3.2 void FilterOutputStream::flush ( ) `[virtual]`

Flushes this output stream and forces any buffered output bytes to be written out to the stream.

The `flush` method of `FilterOutputStream` calls the `flush` method of its underlying output stream.

Reimplemented from OutputStream.

Reimplemented in BufferedOutputStream.

Definition at line 40 of file FilterOutputStream.cpp.

4.15.3.3 void FilterOutputStream::write (unsigned char *b*) `[virtual]`

Writes the specified `unsigned char` to this output stream.

The `write` method of `FilterOutputStream` calls the `write` method of its underlying output stream, that is, it performs `out->write(b)`.

Implements the abstract `write` method of `OutputStream`.

Parameters

b	the `unsigned char`.

Implements OutputStream.

Reimplemented in BufferedOutputStream.

Definition at line 28 of file FilterOutputStream.cpp.

4.15.3.4 void FilterOutputStream::write (unsigned char ∗ *b*, int *len*) `[virtual]`

Writes `len` bytes to this output stream.

The `write` method of `FilterOutputStream` calls its `write` method of two arguments with the arguments `b` and <codelen.

Parameters

b	the data to be written.
len	the length

Reimplemented from OutputStream.

Reimplemented in BufferedOutputStream.

Definition at line 32 of file FilterOutputStream.cpp.

4.15.3.5 void FilterOutputStream::write (unsigned char ∗ *b*, int *off*, int *len*) `[virtual]`

Writes `len` bytes from the specified `unsigned char` array starting at offset `off` to this output stream.

Parameters

b	the data.
off	the start offset in the data.
len	the number of bytes to write.

Reimplemented from OutputStream.

Reimplemented in BufferedOutputStream.

Definition at line 36 of file FilterOutputStream.cpp.

4.15.4 Member Data Documentation

4.15.4.1 OutputStream∗ FilterOutputStream::out `[protected]`

The underlying output stream to be filtered.

Definition at line 30 of file FilterOutputStream.h.

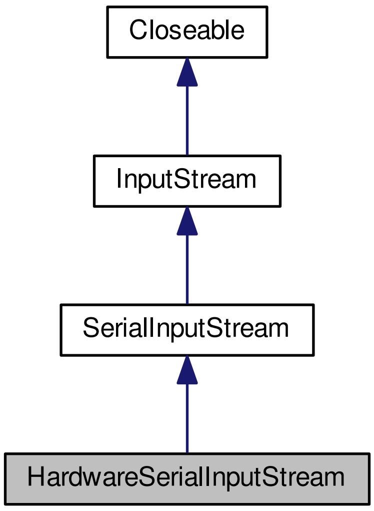
The documentation for this class was generated from the following files:

- FilterOutputStream.h
- FilterOutputStream.cpp

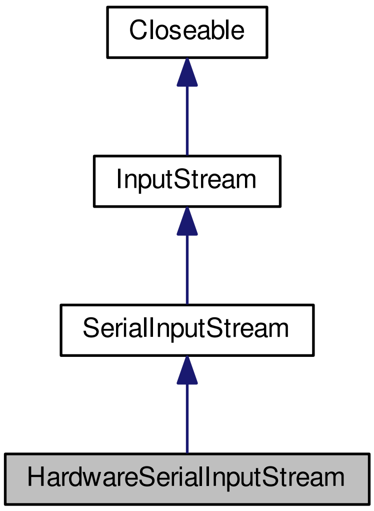
4.16 HardwareSerialInputStream Class Reference

`#include <HardwareSerialInputStream.h>`

Inheritance diagram for HardwareSerialInputStream:



Collaboration diagram for HardwareSerialInputStream:



Public Member Functions

- HardwareSerialInputStream (unsigned int boudRate)
- virtual int available ()
- virtual int read ()

4.16.1 Detailed Description

Arduino IO.

HardwareSerialInputStream

A HardwareSerialInputStream obtains input bytes from a serial port.

Definition at line 16 of file HardwareSerialInputStream.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 HardwareSerialInputStream::HardwareSerialInputStream (unsigned int *boudRate*)

Public constructor.

Parameters

boudRate	

Definition at line 14 of file HardwareSerialInputStream.cpp.

4.16.3 Member Function Documentation

4.16.3.1 int HardwareSerialInputStream::available ( ) `[virtual]`

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Reimplemented from InputStream.

Definition at line 18 of file HardwareSerialInputStream.cpp.

4.16.3.2 int HardwareSerialInputStream::read ( ) `[virtual]`

Reads the next unsigned char of data from the input stream.

Implements InputStream.

Definition at line 22 of file HardwareSerialInputStream.cpp.

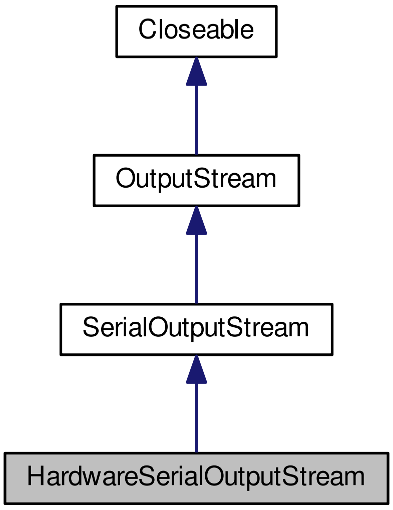
The documentation for this class was generated from the following files:

- HardwareSerialInputStream.h
- HardwareSerialInputStream.cpp

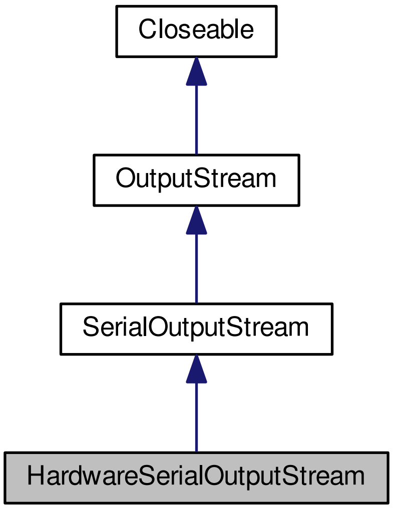
4.17 HardwareSerialOutputStream Class Reference

```
#include <HardwareSerialOutputStream.h>
```

Inheritance diagram for HardwareSerialOutputStream:



Collaboration diagram for HardwareSerialOutputStream:



Public Member Functions

- HardwareSerialOutputStream (unsigned int boudRate)
- virtual void write (unsigned char b)

4.17.1 Detailed Description

Arduino IO.

HardwareSerialOutputStream

A software serial output stream is a output stream to write in a serial port.

Definition at line 16 of file HardwareSerialOutputStream.h.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 HardwareSerialOutputStream::HardwareSerialOutputStream (unsigned int *boudRate*)

Public constructor.

Parameters

boudRate	

Definition at line 14 of file HardwareSerialOutputStream.cpp.

4.17.3 Member Function Documentation

4.17.3.1 void HardwareSerialOutputStream::write (unsigned char *b*) `[virtual]`

Writes the specified unsigned char to this output stream.

Implements OutputStream.

Definition at line 18 of file HardwareSerialOutputStream.cpp.

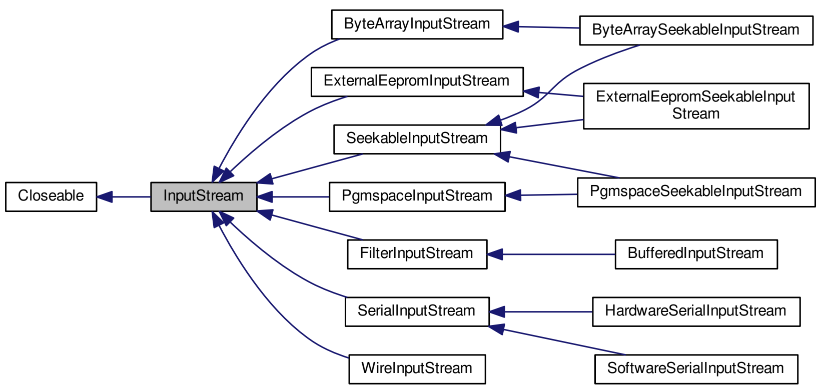
The documentation for this class was generated from the following files:

- HardwareSerialOutputStream.h
- HardwareSerialOutputStream.cpp

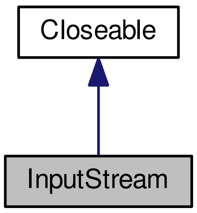
4.18 InputStream Class Reference

`#include <InputStream.h>`

Inheritance diagram for InputStream:


ByteArrayInputStream
ByteArraySeekableInputStream
ExternalEepromInputStream
ExternalEepromSeekableInput Stream
SeekableInputStream
Closeable
InputStream
PgmspaceInputStream
PgmspaceSeekableInputStream
FilterInputStream
BufferedInputStream
SerialInputStream
HardwareSerialInputStream
WireInputStream
SoftwareSerialInputStream

Collaboration diagram for InputStream:



Public Member Functions

- virtual int available ()
- virtual void close ()
- virtual void mark ()
- virtual bool markSupported ()
- virtual int read ()=0
- virtual int read (unsigned char ∗b, int len)
- virtual int read (unsigned char ∗b, int off, int len)
- virtual void reset ()
- virtual unsigned int skip (unsigned int n)

4.18.1 Detailed Description

Arduino IO.

InputStream

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of InputStream must always provide a method that returns the next unsigned char of input.

Definition at line 18 of file InputStream.h.

4.18.2 Member Function Documentation

4.18.2.1 int InputStream::available ( ) [virtual]

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Reimplemented in BufferedInputStream, FilterInputStream, ExternalEepromInputStream, ByteArrayInputStream, PgmspaceInputStream, SoftwareSerialInputStream, WireInputStream, and HardwareSerialInputStream.

Definition at line 18 of file InputStream.cpp.

4.18.2.2 void InputStream::close ( ) [virtual]

Closes this input stream and releases any system resources associated with the stream.

Implements Closeable.

Reimplemented in BufferedInputStream, and FilterInputStream.

Definition at line 22 of file InputStream.cpp.

4.18.2.3 void InputStream::mark ( ) [virtual]

Marks the current position in this input stream.

Reimplemented in BufferedInputStream, FilterInputStream, ExternalEepromInputStream, ByteArrayInputStream, and PgmspaceInputStream.

Definition at line 25 of file InputStream.cpp.

4.18.2.4 bool InputStream::markSupported ( ) [virtual]

Tests if this input stream supports the mark and reset methods.

Reimplemented in FilterInputStream, BufferedInputStream, ExternalEepromInputStream, ByteArrayInputStream, and PgmspaceInputStream.

Definition at line 28 of file InputStream.cpp.

4.18.2.5 virtual int InputStream::read ( ) [pure virtual]

Reads the next unsigned char of data from the input stream.

Implemented in BufferedInputStream, ExternalEepromInputStream, ByteArrayInputStream, PgmspaceInputStream, FilterInputStream, SoftwareSerialInputStream, WireInputStream, and HardwareSerialInputStream.

4.18.2.6 int InputStream::read (unsigned char ∗ *b*, int *len*) [virtual]

Reads some number of bytes from the input stream and stores them into the buffer array b.

Reimplemented in BufferedInputStream, and FilterInputStream.

Definition at line 32 of file InputStream.cpp.

4.18.2.7 int InputStream::read (unsigned char ∗ *b*, int *off*, int *len*) [virtual]

Writes len of bytes into the stream.

Parameters

b	
off	
len	

Returns

Reimplemented in BufferedInputStream, FilterInputStream, ExternalEepromInputStream, and WireInputStream.

Definition at line 36 of file InputStream.cpp.

4.18.2.8 void InputStream::reset ( ) [virtual]

Repositions this stream to the position at the time the mark method was last called on this input stream.

Reimplemented in BufferedInputStream, FilterInputStream, ExternalEepromInputStream, ByteArrayInputStream, and PgmspaceInputStream.

Definition at line 56 of file InputStream.cpp.

4.18.2.9 unsigned int InputStream::skip (unsigned int *n*) [virtual]

Skips over and discards n bytes of data from this input stream.

Reimplemented in BufferedInputStream, and FilterInputStream.

Definition at line 59 of file InputStream.cpp.

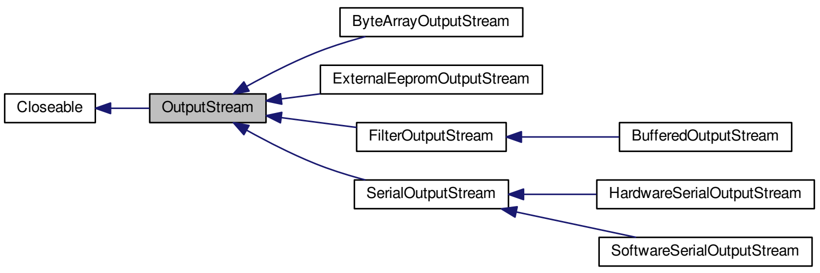
The documentation for this class was generated from the following files:

- InputStream.h
- InputStream.cpp

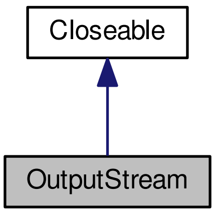
4.19 OutputStream Class Reference

```
#include <OutputStream.h>
```

Inheritance diagram for OutputStream:



Collaboration diagram for OutputStream:



Public Member Functions

- virtual void close ()
- virtual void flush ()
- virtual void write (unsigned char b)=0
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char ∗b, int off, int len)

4.19.1 Detailed Description

Arduino IO.

OutputStream

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Applications that need to define a subclass of `OutputStream` must always provide at least a method that writes one unsigned char of output.

Definition at line 20 of file OutputStream.h.

4.19.2 Member Function Documentation

4.19.2.1 void OutputStream::close ( ) [virtual]

Closes this output stream and releases any system resources associated with this stream.

Implements Closeable.

Reimplemented in FilterOutputStream, and BufferedOutputStream.

Definition at line 34 of file OutputStream.cpp.

4.19.2.2 void OutputStream::flush ( ) [virtual]

Flushes this output stream and forces any buffered output bytes to be written out.

Reimplemented in BufferedOutputStream, and FilterOutputStream.

Definition at line 31 of file OutputStream.cpp.

4.19.2.3 virtual void OutputStream::write (unsigned char *b*) [pure virtual]

Writes the specified unsigned char to this output stream.

Implemented in ByteArrayOutputStream, BufferedOutputStream, FilterOutputStream, ExternalEepromOutput↵Stream, SoftwareSerialOutputStream, and HardwareSerialOutputStream.

4.19.2.4 void OutputStream::write (unsigned char ∗ *b*, int *len*) [virtual]

Writes len bytes from the specified unsigned char array to this output stream.

Parameters

b	
len	

Reimplemented in BufferedOutputStream, and FilterOutputStream.

Definition at line 18 of file OutputStream.cpp.

4.19.2.5 void OutputStream::write (unsigned char ∗ *b*, int *off*, int *len*) [virtual]

Writes len bytes from the specified unsigned char array starting at offset off to this output stream.

Parameters

b	
off	
len	

Reimplemented in BufferedOutputStream, FilterOutputStream, and ExternalEepromOutputStream.

Definition at line 22 of file OutputStream.cpp.

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

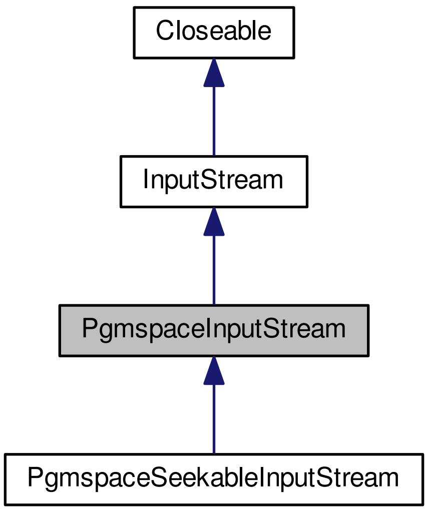
- OutputStream.h

- OutputStream.cpp

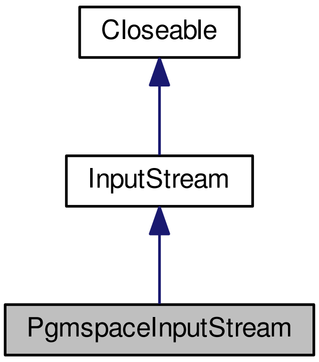
4.20 PgmspaceInputStream Class Reference

`#include <PgmspaceInputStream.h>`

Inheritance diagram for PgmspaceInputStream:



Collaboration diagram for PgmspaceInputStream:



Public Member Functions

- PgmspaceInputStream (char PROGMEM ∗buf, unsigned int count)

- virtual int available ()
- virtual void mark ()
- virtual bool markSupported ()
- virtual int read ()
- virtual void reset ()

Protected Attributes

- char PROGMEM ∗ buf
- unsigned int count
- unsigned int pos
- unsigned int markpos

4.20.1 Detailed Description

Arduino IO.

PgmspaceInputStream

A PgmspaceInputStream contains an internal buffer that contains bytes that may be read from the stream mapped to an part of the pgmspace.

Definition at line 16 of file PgmspaceInputStream.h.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 PgmspaceInputStream::PgmspaceInputStream (char PROGMEM ∗ *buf,* unsigned int *count*) [explicit]

Definition at line 15 of file PgmspaceInputStream.cpp.

4.20.3 Member Function Documentation

4.20.3.1 int PgmspaceInputStream::available ( ) [virtual]

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Returns

Reimplemented from InputStream.

Definition at line 20 of file PgmspaceInputStream.cpp.

4.20.3.2 void PgmspaceInputStream::mark ( ) [virtual]

Marks the current position in this input stream.

Reimplemented from InputStream.

Definition at line 27 of file PgmspaceInputStream.cpp.

4.20.3.3 bool PgmspaceInputStream::markSupported ( ) [virtual]

Tests if this input stream supports the mark and reset methods.

Returns

Reimplemented from InputStream.

Definition at line 31 of file PgmspaceInputStream.cpp.

4.20.3.4 int PgmspaceInputStream::read ( ) `[virtual]`

Reads the next unsigned char of data from the input stream.

Returns

Implements InputStream.

Definition at line 35 of file PgmspaceInputStream.cpp.

4.20.3.5 void PgmspaceInputStream::reset ( ) `[virtual]`

Repositions this stream to the position at the time the mark method was last called on this input stream.

Reimplemented from InputStream.

Definition at line 39 of file PgmspaceInputStream.cpp.

4.20.4 Member Data Documentation

4.20.4.1 char PROGMEM∗ PgmspaceInputStream::buf `[protected]`

Definition at line 22 of file PgmspaceInputStream.h.

4.20.4.2 unsigned int PgmspaceInputStream::count `[protected]`

Definition at line 27 of file PgmspaceInputStream.h.

4.20.4.3 unsigned int PgmspaceInputStream::markpos `[protected]`

Definition at line 37 of file PgmspaceInputStream.h.

4.20.4.4 unsigned int PgmspaceInputStream::pos `[protected]`

Definition at line 32 of file PgmspaceInputStream.h.

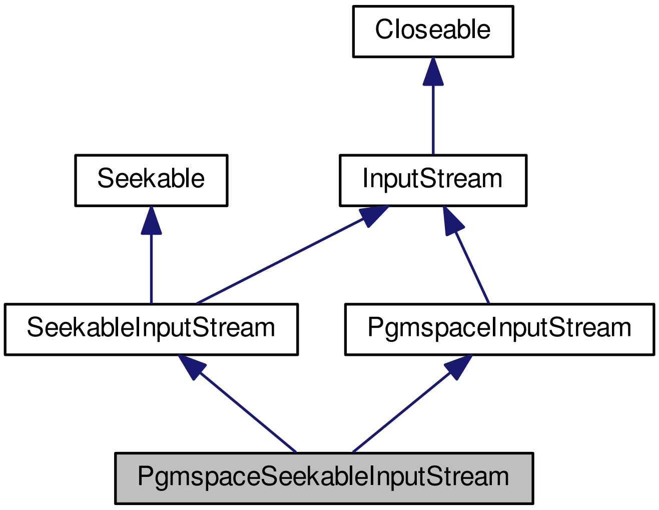
The documentation for this class was generated from the following files:

- PgmspaceInputStream.h
- PgmspaceInputStream.cpp

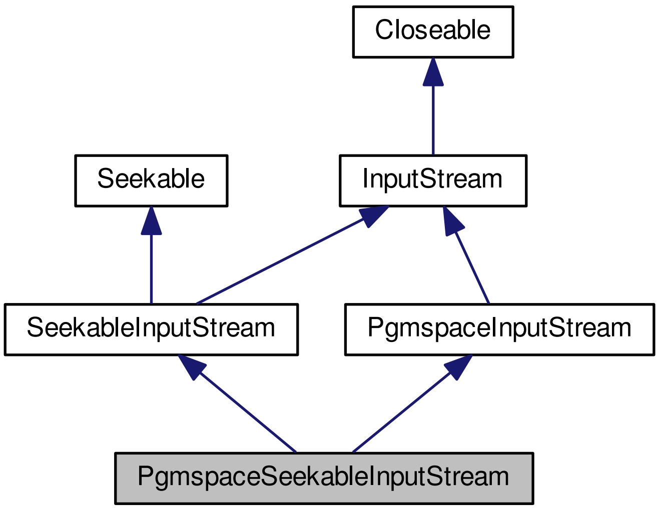
4.21 PgmspaceSeekableInputStream Class Reference

```
#include <PgmspaceSeekableInputStream.h>
```

Inheritance diagram for PgmspaceSeekableInputStream:



Collaboration diagram for PgmspaceSeekableInputStream:



Public Member Functions

- PgmspaceSeekableInputStream (char PROGMEM ∗buf, unsigned int count)
- virtual void seek (unsigned int pos)

Additional Inherited Members

4.21.1 Detailed Description

Arduino IO.

PgmspaceSeekableInputStream

A PgmspaceSeekableInputStream obtains input bytes from a resource in a file system that implements Seekable↩ InputStream interface.

Definition at line 16 of file PgmspaceSeekableInputStream.h.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 PgmspaceSeekableInputStream::PgmspaceSeekableInputStream (char PROGMEM * *buf*, unsigned int *count*)

Definition at line 15 of file PgmspaceSeekableInputStream.cpp.

4.21.3 Member Function Documentation

4.21.3.1 void PgmspaceSeekableInputStream::seek (unsigned int *pos*) `[virtual]`

Implements Seekable.

Definition at line 18 of file PgmspaceSeekableInputStream.cpp.

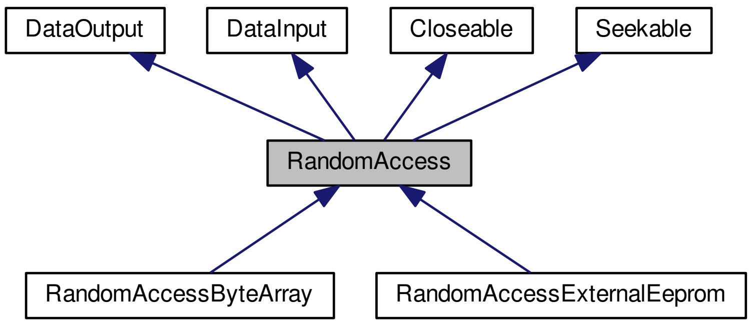
The documentation for this class was generated from the following files:

- PgmspaceSeekableInputStream.h
- PgmspaceSeekableInputStream.cpp

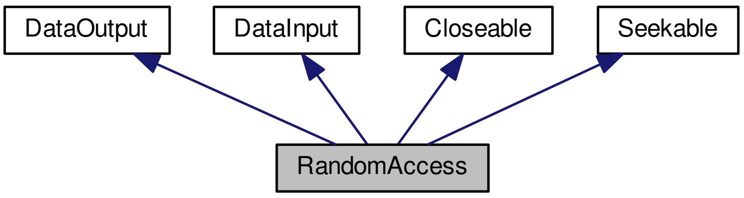
4.22 RandomAccess Class Reference

`#include <RandomAccess.h>`

Inheritance diagram for RandomAccess:



Collaboration diagram for RandomAccess:



Additional Inherited Members

4.22.1 Detailed Description

Araduino IO.

RandomAccess

Interface derived from DataInput, DataOutput, Closeable and Seekable.

Definition at line 17 of file RandomAccess.h.

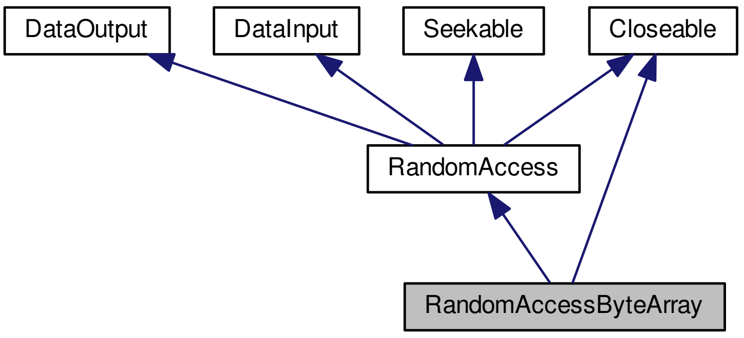
The documentation for this class was generated from the following file:

- RandomAccess.h

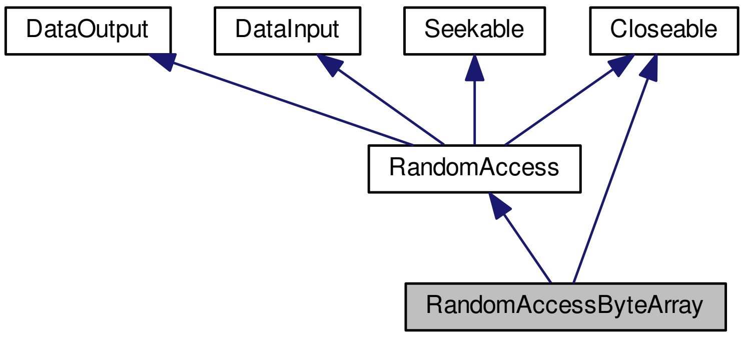
4.23 RandomAccessByteArray Class Reference

```
#include <RandomAccessByteArray.h>
```

Inheritance diagram for RandomAccessByteArray:



Collaboration diagram for RandomAccessByteArray:



Public Member Functions

- RandomAccessByteArray (unsigned char ∗buf, unsigned int count)
- virtual void seek (unsigned int pos)
- unsigned int length ()
- virtual void close ()
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char b)
- virtual void writeByte (unsigned char b)
- virtual void writeBytes (unsigned char ∗b, int len)
- virtual void writeBoolean (bool v)
- virtual void writeChar (char c)
- virtual void writeUnsignedChar (unsigned char c)
- virtual void writeInt (int v)
- virtual void writeUnsignedInt (unsigned int v)
- virtual void writeWord (word v)
- virtual void writeLong (long v)
- virtual void writeUnsignedLong (unsigned long v)
- virtual void writeFloat (float v)
- virtual void writeDouble (double v)
- virtual unsigned char readByte ()
- virtual bool readBoolean ()
- virtual char readChar ()
- virtual unsigned char readUnsignedChar ()
- virtual int readInt ()
- virtual unsigned int readUnsignedInt ()
- virtual word readWord ()
- virtual long readLong ()
- virtual unsigned long readUnsignedLong ()
- virtual float readFloat ()
- virtual double readDouble ()
- virtual void readFully (unsigned char ∗b, int len)
- virtual unsigned int skipBytes (unsigned int n)

Private Attributes

- unsigned char ∗ buf
- unsigned int count
- unsigned int pos

4.23.1 Detailed Description

Araduino IO.

RandomAccessByteArray

Instances of this class support both reading and writing to a random access unsigned char array.

Definition at line 16 of file RandomAccessByteArray.h.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 RandomAccessByteArray::RandomAccessByteArray (unsigned char ∗ *buf,* unsigned int *count*)

Public constructor.

Parameters

buf	The unsigned char array.
count	The size of such unsigned char array.

Definition at line 15 of file RandomAccessByteArray.cpp.

4.23.3 Member Function Documentation

4.23.3.1 void RandomAccessByteArray::close ( ) [virtual]

Closing a unsigned char array has no effect.

Implements Closeable.

Definition at line 29 of file RandomAccessByteArray.cpp.

4.23.3.2 unsigned int RandomAccessByteArray::length ( )

Returns the length of the stream.

Returns

The length.

Definition at line 21 of file RandomAccessByteArray.cpp.

4.23.3.3 bool RandomAccessByteArray::readBoolean ( ) [virtual]

Reads a bool from the stream.

Returns

bool

Implements DataInput.

Definition at line 98 of file RandomAccessByteArray.cpp.

4.23.3.4 unsigned char RandomAccessByteArray::readByte ( ) [virtual]

Reads a unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 94 of file RandomAccessByteArray.cpp.

4.23.3.5 char RandomAccessByteArray::readChar ( ) [virtual]

Reads a char from the stream.

Returns

char

Implements DataInput.

Definition at line 102 of file RandomAccessByteArray.cpp.

4.23.3.6 double RandomAccessByteArray::readDouble ( ) [virtual]

Reads a double from the stream.

Returns

double

Implements DataInput.

Definition at line 146 of file RandomAccessByteArray.cpp.

4.23.3.7 float RandomAccessByteArray::readFloat ( ) [virtual]

Reads a float from the stream.

Returns

float

Implements DataInput.

Definition at line 142 of file RandomAccessByteArray.cpp.

4.23.3.8 void RandomAccessByteArray::readFully (unsigned char ∗ *b*, int *len*) [virtual]

Reads a array of bytes from the stream.

Parameters

b	
len	

Implements DataInput.

Definition at line 150 of file RandomAccessByteArray.cpp.

4.23.3.9 int RandomAccessByteArray::readInt ( ) [virtual]

Reads an int from the stream.

Returns

int

Implements DataInput.

Definition at line 110 of file RandomAccessByteArray.cpp.

4.23.3.10 long RandomAccessByteArray::readLong ( ) `[virtual]`

Reads a long from the stream.

Returns

long

Implements DataInput.

Definition at line 126 of file RandomAccessByteArray.cpp.

4.23.3.11 unsigned char RandomAccessByteArray::readUnsignedChar ( ) `[virtual]`

Reads an unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 106 of file RandomAccessByteArray.cpp.

4.23.3.12 unsigned int RandomAccessByteArray::readUnsignedInt ( ) `[virtual]`

Reads an unsigned int from the stream.

Returns

unsigned int

Implements DataInput.

Definition at line 118 of file RandomAccessByteArray.cpp.

4.23.3.13 unsigned long RandomAccessByteArray::readUnsignedLong ( ) `[virtual]`

Reads a unsigned long from the stream.

Returns

unsigned long

Implements DataInput.

Definition at line 138 of file RandomAccessByteArray.cpp.

4.23.3.14 word RandomAccessByteArray::readWord ( ) `[virtual]`

Reads a word from the stream.

Returns

word

Implements DataInput.

Definition at line 122 of file RandomAccessByteArray.cpp.

4.23.3.15 void RandomAccessByteArray::seek (unsigned int *pos*) [virtual]

Seeks the stream at the position.

Parameters

pos	The position.

Implements Seekable.

Definition at line 25 of file RandomAccessByteArray.cpp.

4.23.3.16 unsigned int RandomAccessByteArray::skipBytes (unsigned int *n*) [virtual]

Skips n bytes of the stream.

Parameters

n	

Returns

unsigned int The number of skipped bytes.

Implements DataInput.

Definition at line 156 of file RandomAccessByteArray.cpp.

4.23.3.17 void RandomAccessByteArray::write (unsigned char ∗ *b*, int *len*) [virtual]

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 32 of file RandomAccessByteArray.cpp.

4.23.3.18 void RandomAccessByteArray::write (unsigned char *b*) [virtual]

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 36 of file RandomAccessByteArray.cpp.

4.23.3.19 void RandomAccessByteArray::writeBoolean (bool *v*) [virtual]

Writes a bool into the stream.

Parameters

v	The bool to be written.

Implements DataOutput.

Definition at line 50 of file RandomAccessByteArray.cpp.

4.23.3.20 void RandomAccessByteArray::writeByte (unsigned char *b*) [virtual]

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 40 of file RandomAccessByteArray.cpp.

4.23.3.21 void RandomAccessByteArray::writeBytes (unsigned char $*$ *b*, int *len*) [virtual]

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 44 of file RandomAccessByteArray.cpp.

4.23.3.22 void RandomAccessByteArray::writeChar (char *c*) [virtual]

Writes a char into the stream.

Parameters

c	The char to be written.

Implements DataOutput.

Definition at line 54 of file RandomAccessByteArray.cpp.

4.23.3.23 void RandomAccessByteArray::writeDouble (double *v*) [virtual]

Writes a double into the stream.

Parameters

v	The double to be written.

Implements DataOutput.

Definition at line 90 of file RandomAccessByteArray.cpp.

4.23.3.24 void RandomAccessByteArray::writeFloat (float *v*) [virtual]

Writes a float into the stream.

Parameters

v	The float to be written.

Implements DataOutput.

Definition at line 86 of file RandomAccessByteArray.cpp.

4.23.3.25 void RandomAccessByteArray::writeInt (int *v*) [virtual]

Writes an int into the stream.

Parameters

v	The int to be written.

Implements DataOutput.

Definition at line 62 of file RandomAccessByteArray.cpp.

4.23.3.26 void RandomAccessByteArray::writeLong (long *v*) [virtual]

Writes a long into the stream.

Parameters

v	The long to be written.

Implements DataOutput.

Definition at line 75 of file RandomAccessByteArray.cpp.

4.23.3.27 void RandomAccessByteArray::writeUnsignedChar (unsigned char *c*) [virtual]

Writes an unsigned char into the stream.

Parameters

c	The unsigned char to be written.

Implements DataOutput.

Definition at line 58 of file RandomAccessByteArray.cpp.

4.23.3.28 void RandomAccessByteArray::writeUnsignedInt (unsigned int *v*) [virtual]

Writes an unsigned int into the stream.

Parameters

v	The unsigned int to be written.

Implements DataOutput.

Definition at line 67 of file RandomAccessByteArray.cpp.

4.23.3.29 void RandomAccessByteArray::writeUnsignedLong (unsigned long *v*) [virtual]

Writes a unsigned long into the stream.

Parameters

v	The unsigned long to be written.

Implements DataOutput.

Definition at line 82 of file RandomAccessByteArray.cpp.

4.23.3.30 void RandomAccessByteArray::writeWord (word *v*) [virtual]

Writes a word into the stream.

Parameters

v	The word to be written.

Implements DataOutput.

Definition at line 71 of file RandomAccessByteArray.cpp.

4.23.4 Member Data Documentation

4.23.4.1 unsigned char* RandomAccessByteArray::buf [private]

Buffer used to work.

Definition at line 21 of file RandomAccessByteArray.h.

4.23.4.2 unsigned int RandomAccessByteArray::count [private]

Buffer size.

Definition at line 26 of file RandomAccessByteArray.h.

4.23.4.3 unsigned int RandomAccessByteArray::pos `[private]`

Current position.

Definition at line 31 of file RandomAccessByteArray.h.

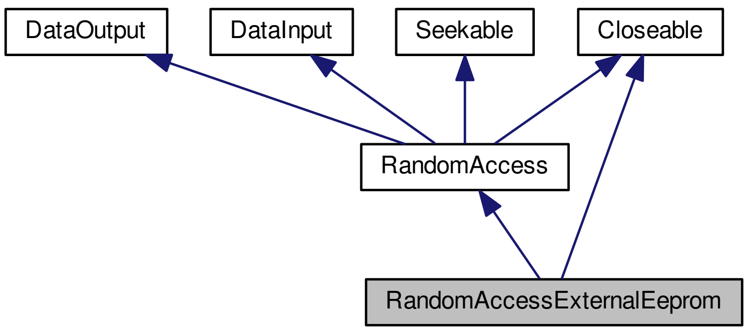
The documentation for this class was generated from the following files:

- RandomAccessByteArray.h
- RandomAccessByteArray.cpp

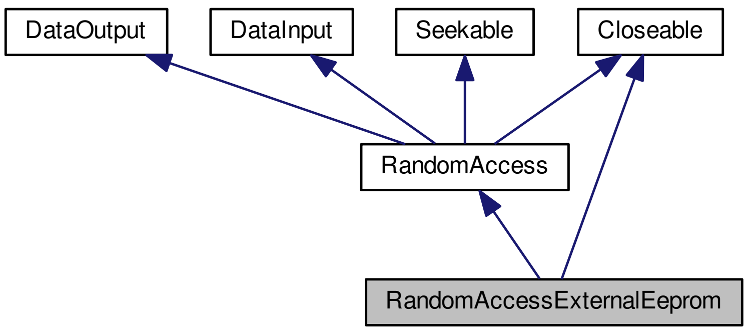
4.24 RandomAccessExternalEeprom Class Reference

`#include <RandomAccessExternalEeprom.h>`

Inheritance diagram for RandomAccessExternalEeprom:



Collaboration diagram for RandomAccessExternalEeprom:



Public Member Functions

- RandomAccessExternalEeprom (ExternalEeprom ∗externalEeprom)
- virtual void seek (unsigned int pos)
- unsigned int length ()
- virtual void close ()
- virtual void write (unsigned char ∗b, int len)
- virtual void write (unsigned char b)
- virtual void writeByte (unsigned char b)
- virtual void writeBytes (unsigned char ∗b, int len)
- virtual void writeBoolean (bool v)
- virtual void writeChar (char c)
- virtual void writeUnsignedChar (unsigned char c)
- virtual void writeInt (int v)
- virtual void writeUnsignedInt (unsigned int v)
- virtual void writeWord (word v)
- virtual void writeLong (long v)
- virtual void writeUnsignedLong (unsigned long v)
- virtual void writeFloat (float v)
- virtual void writeDouble (double v)
- virtual unsigned char readByte ()
- virtual bool readBoolean ()
- virtual char readChar ()
- virtual unsigned char readUnsignedChar ()
- virtual int readInt ()
- virtual unsigned int readUnsignedInt ()
- virtual word readWord ()
- virtual long readLong ()
- virtual unsigned long readUnsignedLong ()
- virtual float readFloat ()
- virtual double readDouble ()
- virtual void readFully (unsigned char ∗b, int len)
- virtual unsigned int skipBytes (unsigned int n)

Private Attributes

- ExternalEeprom ∗ externalEeprom
- unsigned int pos

4.24.1 Detailed Description

Araduino IO.

RandomAccessExternalEeprom

Instances of this class support both reading and writing to a random access externalEeprom. A random access externalEeprom behaves like a large array of bytes stored in the externalEeprom system.

Definition at line 19 of file RandomAccessExternalEeprom.h.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 RandomAccessExternalEeprom::RandomAccessExternalEeprom (ExternalEeprom ∗ *externalEeprom*)

Public constructor.

Parameters

externalEeprom	The external eeprom instance to be used.

Definition at line 17 of file RandomAccessExternalEeprom.cpp.

4.24.3 Member Function Documentation

4.24.3.1 void RandomAccessExternalEeprom::close ( ) `[virtual]`

Closing a external eeprom has no effect.

Implements Closeable.

Definition at line 31 of file RandomAccessExternalEeprom.cpp.

4.24.3.2 unsigned int RandomAccessExternalEeprom::length ( )

Returns the length of the stream.

Returns

The length.

Definition at line 23 of file RandomAccessExternalEeprom.cpp.

4.24.3.3 bool RandomAccessExternalEeprom::readBoolean ( ) `[virtual]`

Reads a bool from the stream.

Returns

bool

Implements DataInput.

Definition at line 100 of file RandomAccessExternalEeprom.cpp.

4.24.3.4 unsigned char RandomAccessExternalEeprom::readByte ( ) `[virtual]`

Reads a unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 96 of file RandomAccessExternalEeprom.cpp.

4.24.3.5 char RandomAccessExternalEeprom::readChar ( ) `[virtual]`

Reads a char from the stream.

Returns

char

Implements DataInput.

Definition at line 104 of file RandomAccessExternalEeprom.cpp.

4.24.3.6 double RandomAccessExternalEeprom::readDouble ( ) [virtual]

Reads a double from the stream.

Returns

double

Implements DataInput.

Definition at line 148 of file RandomAccessExternalEeprom.cpp.

4.24.3.7 float RandomAccessExternalEeprom::readFloat ( ) [virtual]

Reads a float from the stream.

Returns

float

Implements DataInput.

Definition at line 144 of file RandomAccessExternalEeprom.cpp.

4.24.3.8 void RandomAccessExternalEeprom::readFully (unsigned char ∗ *b*, int *len*) [virtual]

Reads a array of bytes from the stream.

Parameters

b	
len	

Implements DataInput.

Definition at line 152 of file RandomAccessExternalEeprom.cpp.

4.24.3.9 int RandomAccessExternalEeprom::readInt ( ) [virtual]

Reads an int from the stream.

Returns

int

Implements DataInput.

Definition at line 112 of file RandomAccessExternalEeprom.cpp.

4.24.3.10 long RandomAccessExternalEeprom::readLong ( ) [virtual]

Reads a long from the stream.

Returns

long

Implements DataInput.

Definition at line 128 of file RandomAccessExternalEeprom.cpp.

4.24.3.11 unsigned char RandomAccessExternalEeprom::readUnsignedChar ( ) [virtual]

Reads an unsigned char from the stream.

Returns

unsigned char

Implements DataInput.

Definition at line 108 of file RandomAccessExternalEeprom.cpp.

4.24.3.12 unsigned int RandomAccessExternalEeprom::readUnsignedInt ( ) [virtual]

Reads an unsigned int from the stream.

Returns

unsigned int

Implements DataInput.

Definition at line 120 of file RandomAccessExternalEeprom.cpp.

4.24.3.13 unsigned long RandomAccessExternalEeprom::readUnsignedLong ( ) [virtual]

Reads a unsigned long from the stream.

Returns

unsigned long

Implements DataInput.

Definition at line 140 of file RandomAccessExternalEeprom.cpp.

4.24.3.14 word RandomAccessExternalEeprom::readWord ( ) [virtual]

Reads a word from the stream.

Returns

word

Implements DataInput.

Definition at line 124 of file RandomAccessExternalEeprom.cpp.

4.24.3.15 void RandomAccessExternalEeprom::seek (unsigned int *pos*) [virtual]

Seeks the stream at the position.

Parameters

pos	The position.

Implements Seekable.

Definition at line 27 of file RandomAccessExternalEeprom.cpp.

4.24.3.16 unsigned int RandomAccessExternalEeprom::skipBytes (unsigned int *n*) [virtual]

Skips n bytes of the stream.

Parameters

n	

Returns

unsigned int The number of skipped bytes.

Implements DataInput.

Definition at line 158 of file RandomAccessExternalEeprom.cpp.

4.24.3.17 void RandomAccessExternalEeprom::write (unsigned char ∗ *b*, int *len*) `[virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 34 of file RandomAccessExternalEeprom.cpp.

4.24.3.18 void RandomAccessExternalEeprom::write (unsigned char *b*) `[virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 38 of file RandomAccessExternalEeprom.cpp.

4.24.3.19 void RandomAccessExternalEeprom::writeBoolean (bool *v*) `[virtual]`

Writes a bool into the stream.

Parameters

v	The bool to be written.

Implements DataOutput.

Definition at line 52 of file RandomAccessExternalEeprom.cpp.

4.24.3.20 void RandomAccessExternalEeprom::writeByte (unsigned char *b*) `[virtual]`

Writes a unsigned char into the stream.

Parameters

b	The unsigned char to be written.

Implements DataOutput.

Definition at line 42 of file RandomAccessExternalEeprom.cpp.

4.24.3.21 void RandomAccessExternalEeprom::writeBytes (unsigned char ∗ *b*, int *len*) `[virtual]`

Writes an array of bytes into the stream.

Parameters

b	The array of bytes.
len	The length of such array.

Implements DataOutput.

Definition at line 46 of file RandomAccessExternalEeprom.cpp.

4.24.3.22 void RandomAccessExternalEeprom::writeChar (char *c*) `[virtual]`

Writes a char into the stream.

Parameters

c	The char to be written.

Implements DataOutput.

Definition at line 56 of file RandomAccessExternalEeprom.cpp.

4.24.3.23 void RandomAccessExternalEeprom::writeDouble (double *v*) `[virtual]`

Writes a double into the stream.

Parameters

v	The double to be written.

Implements DataOutput.

Definition at line 92 of file RandomAccessExternalEeprom.cpp.

4.24.3.24 void RandomAccessExternalEeprom::writeFloat (float *v*) `[virtual]`

Writes a float into the stream.

Parameters

v	The float to be written.

Implements DataOutput.

Definition at line 88 of file RandomAccessExternalEeprom.cpp.

4.24.3.25 void RandomAccessExternalEeprom::writeInt (int *v*) `[virtual]`

Writes an int into the stream.

Parameters

v	The int to be written.

Implements DataOutput.

Definition at line 64 of file RandomAccessExternalEeprom.cpp.

4.24.3.26 void RandomAccessExternalEeprom::writeLong (long *v*) `[virtual]`

Writes a long into the stream.

Parameters

v	The long to be written.

Implements DataOutput.

Definition at line 77 of file RandomAccessExternalEeprom.cpp.

4.24.3.27 void RandomAccessExternalEeprom::writeUnsignedChar (unsigned char *c*) [virtual]

Writes an unsigned char into the stream.

Parameters

c	The unsigned char to be written.

Implements DataOutput.

Definition at line 60 of file RandomAccessExternalEeprom.cpp.

4.24.3.28 void RandomAccessExternalEeprom::writeUnsignedInt (unsigned int *v*) [virtual]

Writes an unsigned int into the stream.

Parameters

v	The unsigned int to be written.

Implements DataOutput.

Definition at line 69 of file RandomAccessExternalEeprom.cpp.

4.24.3.29 void RandomAccessExternalEeprom::writeUnsignedLong (unsigned long *v*) [virtual]

Writes a unsigned long into the stream.

Parameters

v	The unsigned long to be written.

Implements DataOutput.

Definition at line 84 of file RandomAccessExternalEeprom.cpp.

4.24.3.30 void RandomAccessExternalEeprom::writeWord (word *v*) [virtual]

Writes a word into the stream.

Parameters

v	The word to be written.

Implements DataOutput.

Definition at line 73 of file RandomAccessExternalEeprom.cpp.

4.24.4 Member Data Documentation

4.24.4.1 ExternalEeprom* RandomAccessExternalEeprom::externalEeprom [private]

The external eeprom to be used.

Definition at line 24 of file RandomAccessExternalEeprom.h.

4.24.4.2 unsigned int RandomAccessExternalEeprom::pos [private]

Current position.

Definition at line 29 of file RandomAccessExternalEeprom.h.

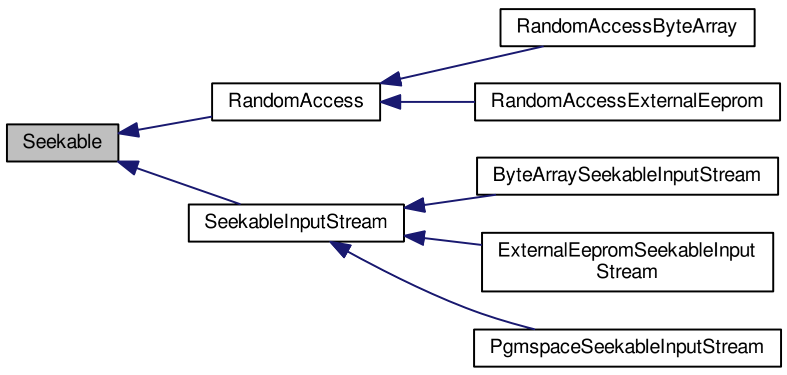
The documentation for this class was generated from the following files:

- RandomAccessExternalEeprom.h
- RandomAccessExternalEeprom.cpp

4.25 Seekable Class Reference

```
#include <Seekable.h>
```

Inheritance diagram for Seekable:


RandomAccessByteArray
RandomAccess
RandomAccessExternalEeprom
Seekable
ByteArraySeekableInputStream
SeekableInputStream
ExternalEepromSeekableInput Stream
PgmspaceSeekableInputStream

Public Member Functions

- virtual void seek (unsigned int pos)=0

4.25.1 Detailed Description

Arduino IO.

Seekable

Definition at line 10 of file Seekable.h.

4.25.2 Member Function Documentation

4.25.2.1 virtual void Seekable::seek (unsigned int *pos*) `[pure virtual]`

Implemented in RandomAccessByteArray, RandomAccessExternalEeprom, ExternalEepromSeekableInput↩ Stream, ByteArraySeekableInputStream, and PgmspaceSeekableInputStream.

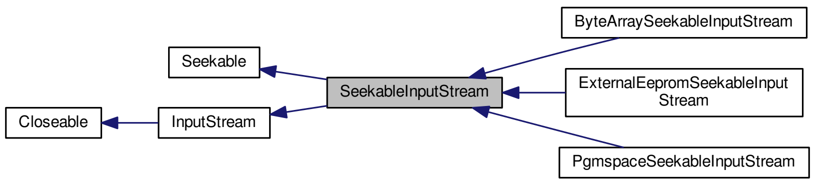
The documentation for this class was generated from the following file:

- Seekable.h

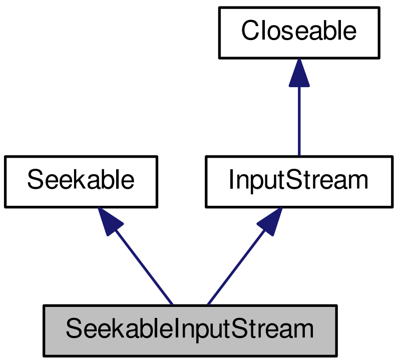
4.26 SeekableInputStream Class Reference

`#include <SeekableInputStream.h>`

Inheritance diagram for SeekableInputStream:


ByteArraySeekableInputStream
Seekable
SeekableInputStream
ExternalEepromSeekableInput Stream
Closeable
InputStream
PgmspaceSeekableInputStream

Collaboration diagram for SeekableInputStream:



Additional Inherited Members

4.26.1 Detailed Description

Arduino IO.

SeekableInputStream

Definition at line 13 of file SeekableInputStream.h.

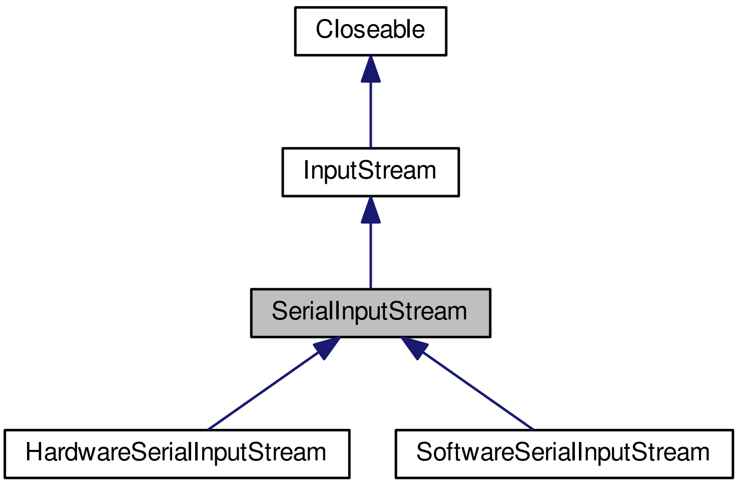
The documentation for this class was generated from the following file:

- SeekableInputStream.h

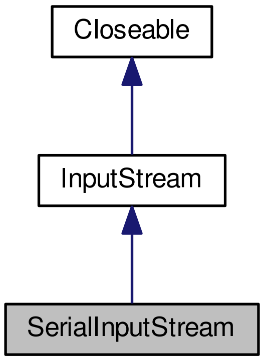
4.27 SerialInputStream Class Reference

```
#include <SerialInputStream.h>
```

Inheritance diagram for SerialInputStream:



Collaboration diagram for SerialInputStream:



Additional Inherited Members

4.27.1 Detailed Description

Arduino IO.

SerialInputStream

A SerialInputStream obtains input bytes from a serial port.

Definition at line 14 of file SerialInputStream.h.

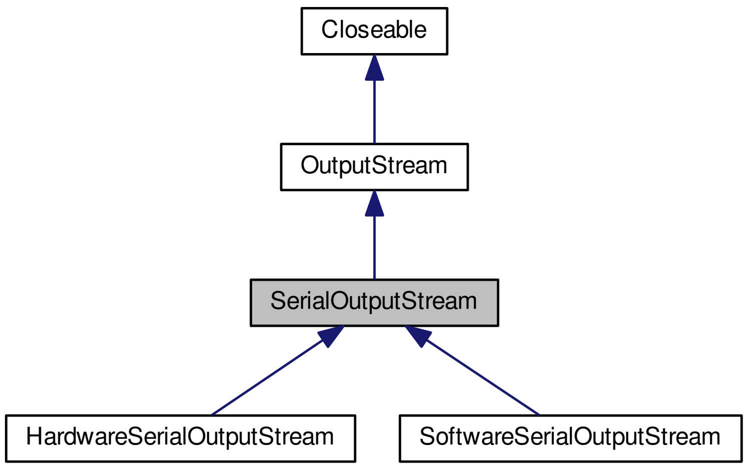
The documentation for this class was generated from the following file:

- SerialInputStream.h

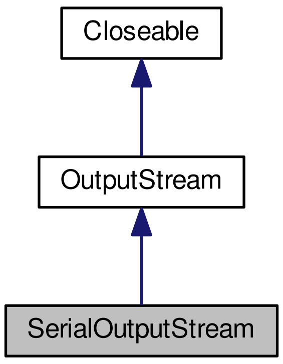
4.28 SerialOutputStream Class Reference

`#include <SerialOutputStream.h>`

Inheritance diagram for SerialOutputStream:



Collaboration diagram for SerialOutputStream:



Additional Inherited Members

4.28.1 Detailed Description

Arduino IO.

SerialOutputStream

A serial output stream is a output stream to write in a serial port.

Definition at line 14 of file SerialOutputStream.h.

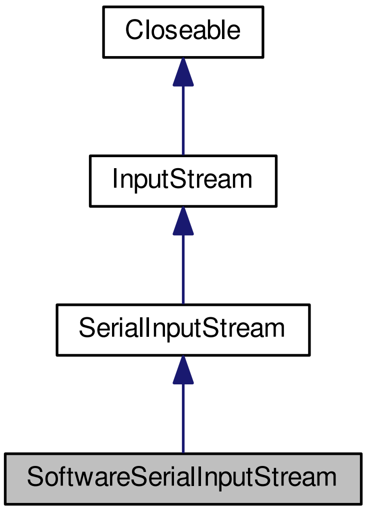
The documentation for this class was generated from the following file:

- SerialOutputStream.h

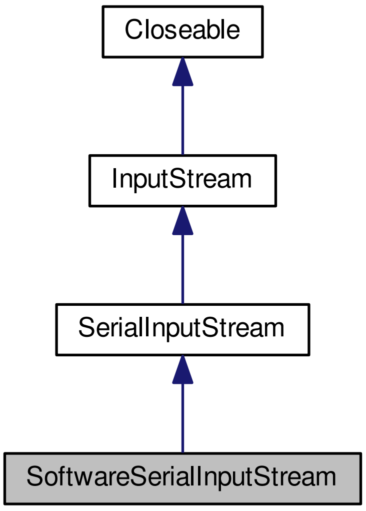
4.29 SoftwareSerialInputStream Class Reference

```
#include <SoftwareSerialInputStream.h>
```

Inheritance diagram for SoftwareSerialInputStream:



Collaboration diagram for SoftwareSerialInputStream:



Public Member Functions

- SoftwareSerialInputStream (SoftwareSerial ∗softwareSerial, unsigned int boudRate)
- virtual int available ()
- virtual int read ()

Protected Attributes

- SoftwareSerial ∗ softwareSerial

4.29.1 Detailed Description

Arduino IO.

SoftwareSerialInputStream

A SoftwareSerialInputStream obtains input bytes from a serial port.

Definition at line 17 of file SoftwareSerialInputStream.h.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 SoftwareSerialInputStream::SoftwareSerialInputStream (SoftwareSerial ∗ *softwareSerial,* unsigned int *boudRate*)

Public constructor.

Parameters

serial	
boudRate	

Definition at line 14 of file SoftwareSerialInputStream.cpp.

4.29.3 Member Function Documentation

4.29.3.1 int SoftwareSerialInputStream::available ( ) `[virtual]`

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Reimplemented from InputStream.

Definition at line 20 of file SoftwareSerialInputStream.cpp.

4.29.3.2 int SoftwareSerialInputStream::read ( ) `[virtual]`

Reads the next unsigned char of data from the input stream.

Implements InputStream.

Definition at line 24 of file SoftwareSerialInputStream.cpp.

4.29.4 Member Data Documentation

4.29.4.1 SoftwareSerial$*$ SoftwareSerialInputStream::softwareSerial `[protected]`

The software serial where the data will be read.

Definition at line 23 of file SoftwareSerialInputStream.h.

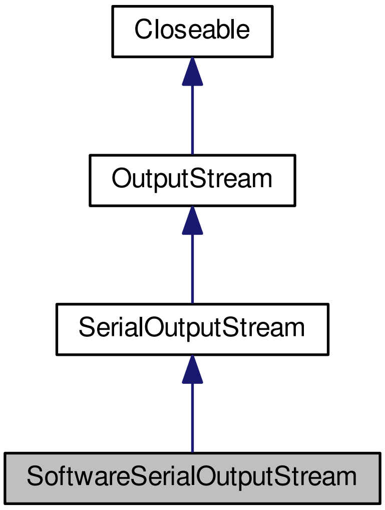
The documentation for this class was generated from the following files:

- SoftwareSerialInputStream.h
- SoftwareSerialInputStream.cpp

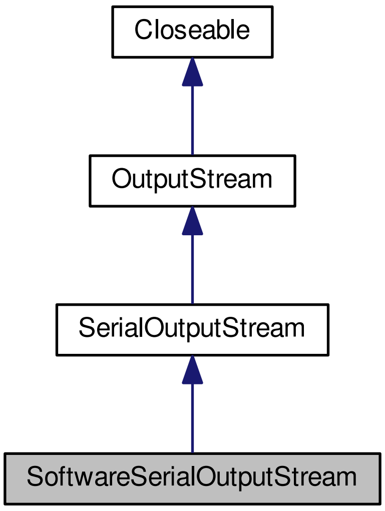
4.30 SoftwareSerialOutputStream Class Reference

```
#include <SoftwareSerialOutputStream.h>
```

Inheritance diagram for SoftwareSerialOutputStream:



Collaboration diagram for SoftwareSerialOutputStream:



Public Member Functions

- SoftwareSerialOutputStream (SoftwareSerial ∗serial, unsigned int boudRate)
- virtual void write (unsigned char b)

Protected Attributes

- SoftwareSerial ∗ softwareSerial

4.30.1 Detailed Description

Arduino IO.

SoftwareSerialOutputStream

A software serial output stream is a output stream to write in a serial port.

Definition at line 17 of file SoftwareSerialOutputStream.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 SoftwareSerialOutputStream::SoftwareSerialOutputStream (SoftwareSerial ∗ *serial,* unsigned int *boudRate*)

Definition at line 14 of file SoftwareSerialOutputStream.cpp.

4.30.3 Member Function Documentation

4.30.3.1 void SoftwareSerialOutputStream::write (unsigned char *b*) `[virtual]`

Writes the specified unsigned char to this output stream.

Implements OutputStream.

Definition at line 20 of file SoftwareSerialOutputStream.cpp.

4.30.4 Member Data Documentation

4.30.4.1 SoftwareSerial∗ SoftwareSerialOutputStream::softwareSerial `[protected]`

Definition at line 23 of file SoftwareSerialOutputStream.h.

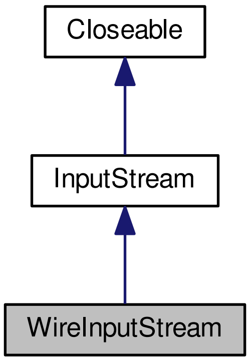
The documentation for this class was generated from the following files:

- SoftwareSerialOutputStream.h
- SoftwareSerialOutputStream.cpp

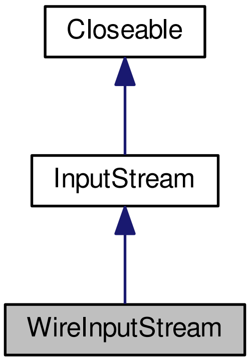
4.31 WireInputStream Class Reference

```
#include <WireInputStream.h>
```

Inheritance diagram for WireInputStream:



Collaboration diagram for WireInputStream:



Public Member Functions

- WireInputStream (unsigned char addredd)
- virtual int available ()
- virtual int read ()
- virtual int read (unsigned char ∗b, int off, int len)

Protected Attributes

- unsigned char address

4.31.1 Detailed Description

Arduino IO.

WireInputStream

A WireInputStream obtains input bytes from the wire bus.

Definition at line 16 of file WireInputStream.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 WireInputStream::WireInputStream (unsigned char *addredd*)

Public constructor.

Parameters

address	

Definition at line 14 of file WireInputStream.cpp.

4.31.3 Member Function Documentation

4.31.3.1 int WireInputStream::available ( ) [virtual]

Returns the number of bytes that can be read(or skipped over) from this input stream without blocking by the next caller of a method for this input stream.

Reimplemented from InputStream.

Definition at line 19 of file WireInputStream.cpp.

4.31.3.2 int WireInputStream::read ( ) [virtual]

Reads the next unsigned char of data from the input stream.

Implements InputStream.

Definition at line 23 of file WireInputStream.cpp.

4.31.3.3 int WireInputStream::read (unsigned char $*$ *b,* int *off,* int *len*) [virtual]

Writes len of bytes into the stream.

Parameters

b	
off	
len	

Returns

Reimplemented from InputStream.

Definition at line 33 of file WireInputStream.cpp.

4.31.4 Member Data Documentation

4.31.4.1 unsigned char WireInputStream::address [protected]

The wire device address.

Definition at line 22 of file WireInputStream.h.

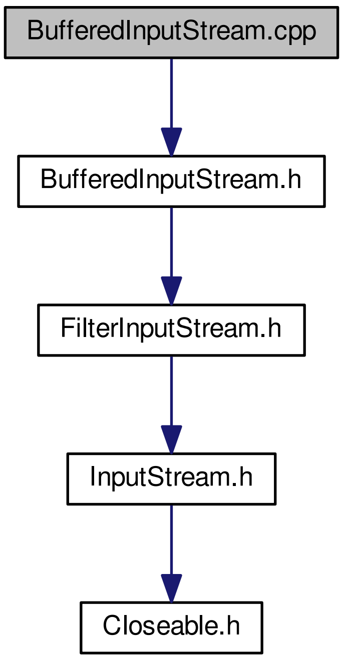
The documentation for this class was generated from the following files:

- WireInputStream.h
- WireInputStream.cpp

5 File Documentation

5.1 BufferedInputStream.cpp File Reference

`#include "BufferedInputStream.h"`
Include dependency graph for BufferedInputStream.cpp:



Macros

- #define __ARDUINO_IO_BUFFERED_INPUT_STREAM_CPP__ 1

5.1.1 Macro Definition Documentation

5.1.1.1 #define __ARDUINO_IO_BUFFERED_INPUT_STREAM_CPP__ 1

Arduino IO.

BufferedInputStream

A `BufferedInputStream` adds functionality to another input stream-namely, the ability to buffer the input and to support the `mark` and `reset` methods. When the `BufferedInputStream` is created, an internal buffer array is passed. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time. The `mark` operation remembers a point in the input stream and the `reset` operation causes all the bytes read since the most recent `mark` operation to be reread before new bytes are taken from the contained input stream.

Definition at line 25 of file BufferedInputStream.cpp.

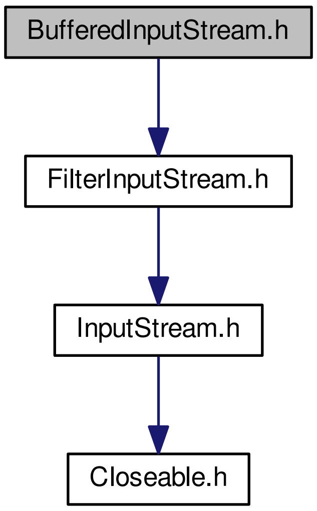
5.2 BufferedInputStream.cpp

```
00001
00024 #ifndef __ARDUINO_IO_BUFFERED_INPUT_STREAM_CPP__
00025 #define __ARDUINO_IO_BUFFERED_INPUT_STREAM_CPP__ 1
00026
00027 #include "BufferedInputStream.h"
00028
00029 BufferedInputStream::BufferedInputStream(
      InputStream* in,
00030         unsigned char* buf, int size) :
00031         FilterInputStream(in), buf(buf) {
00032     this->size = size;
00033     count = 0;
00034     pos = 0;
00035 }
00036
00037 int BufferedInputStream::available() {
00038     return in->available() + (count - pos);
00039 }
00040
00041 void BufferedInputStream::close() {
00042     in->close();
00043 }
00044
00045 void BufferedInputStream::reset() {
00046     if (marked) {
00047         pos = markpos;
00048     }
00049 }
00050
00051 int BufferedInputStream::read(unsigned char* b, int len) {
00052     return read(b, 0, len);
00053 }
00054
00055 int BufferedInputStream::read(unsigned char* b, int off, int len) {
00056     int cnt, available;
00057     available = count - pos;
00058
00059     /*
00060      * The needed data are already in the buffer?
00061      */
00062     if (available >= len) {
00063         for (int i = 0; i < len; i++) {
00064             b[off + i] = buf[pos + i];
00065         }
00066         pos += len;
00067         return len;
00068     }
00069
00070     /*
00071      * The buffer data is not enough, but is necessary.
00072      */
00073     for (int i = 0; i < available; i++) {
00074         b[off + i] = buf[pos + i];
00075     }
00076     marked = false;
00077     pos = 0;
00078     count = 0;
00079
00080     /*
00081      * Reads the rest from the stream.
00082      */
00083     cnt = in->read(b, off + available, len - available);
00084
00085     /*
00086      * Tests if we had enough data.
00087      */
00088     if (cnt < 0) {
00089         return available;
00090     } else if (cnt < (len - available)) {
00091         return available + cnt;
00092     } else {
00093         fill(0);
00094     }
00095     return len;
}

int BufferedInputStream::read() {

    /*
     * Tests if the buffer is completely used.
     */
    if (pos >= count) {
        marked = false;
        fill(0);
        if (count == 0) {
            return -1;
        }
        pos = 0;
    }
    return (int) buf[pos++];
}

void BufferedInputStream::realineBufferContent() {
    int n;
    if (pos > 0) {
        n = count - pos;
        for (int i = 0; i < n; i++) {
            buf[i] = buf[pos + i];
        }
        count -= pos;
        pos = 0;
    }
}

void BufferedInputStream::fill(int startPos) {
    int n, needed;
    needed = size - startPos;
    if (needed <= 0) {
        return;
    }
    n = in->read(buf, startPos, needed);
    if (n > 0) {
        count = startPos + n;
    }
}

void BufferedInputStream::mark() {
    realineBufferContent();
    fill(count);
    markpos = 0;
    marked = true;
}

bool BufferedInputStream::markSupported() {
    return true;
}

unsigned int BufferedInputStream::skip(unsigned int n) {
    unsigned int buffered, skiped;
    buffered = count - pos;
    if (buffered >= n) {
        pos += n;
        return n;
    }
    pos = 0;
    count = 0;
    marked = false;
    skiped = buffered + in->skip(n - buffered);
    return skiped;
}

#endif /* __ARDUINO_IO_BUFFERED_INPUT_STREAM_CPP__ */
```

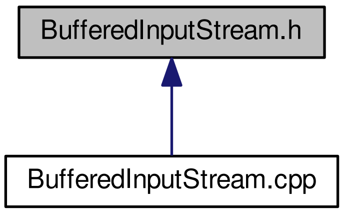
5.3 BufferedInputStream.h File Reference

```
#include <FilterInputStream.h>
```

Include dependency graph for BufferedInputStream.h:



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



Classes

- class BufferedInputStream

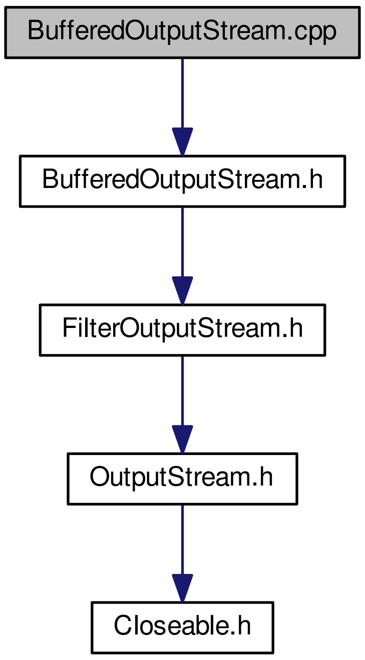
5.4 BufferedInputStream.h

```

#ifndef __ARDUINO_IO_BUFFERED_INPUT_STREAM_H__
#define __ARDUINO_IO_BUFFERED_INPUT_STREAM_H__ 1

#include <FilterInputStream.h>

class BufferedInputStream : public FilterInputStream {

    unsigned int size;

protected:

    unsigned char* buf;

    int count;

    int pos;

    int markpos;

    bool marked;

public:

    BufferedInputStream(InputStream* in, unsigned char* buf, int size);

    virtual int available();

    virtual void close();

    virtual void mark();

    virtual bool markSupported();

    virtual int read();

    virtual int read(unsigned char* b, int len);

    virtual int read(unsigned char* b, int off, int len);

    virtual void reset();

    virtual unsigned int skip(unsigned int n);

private:

    void realineBufferContent();

    void fill(int startPos);
};

#endif /* __ARDUINO_IO_BUFFERED_INPUT_STREAM_H__ */
```

5.5 BufferedOutputStream.cpp File Reference

```
#include "BufferedOutputStream.h"
```

Include dependency graph for BufferedOutputStream.cpp:



Macros

- #define __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_CPP__ 1

5.5.1 Macro Definition Documentation

5.5.1.1 #define __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_CPP__ 1

Arduino IO.

BufferedOutputStream

The class implements a buffered output stream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each unsigned char written.

Definition at line 13 of file BufferedOutputStream.cpp.

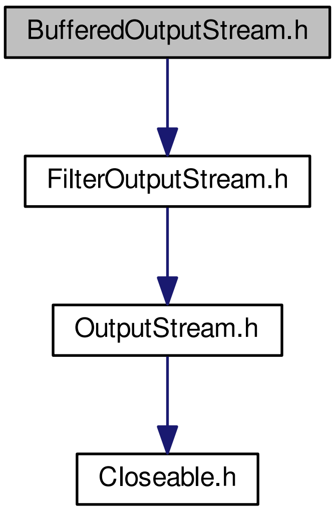
5.6 BufferedOutputStream.cpp

```
00001
00012 #ifndef __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_CPP__
00013 #define __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_CPP__ 1
00014
00015 #include "BufferedOutputStream.h"
00016
00017 BufferedOutputStream::BufferedOutputStream(
      OutputStream* out,
00018         unsigned char* buf, int size) :
00019         FilterOutputStream(out), buf(buf) {
    this->size = size;
    count = 0;
}

void BufferedOutputStream::write(unsigned char b) {
    if (count >= size) {
        flushBuffer();
    }
    buf[count++] = b;
}

void BufferedOutputStream::write(unsigned char* b, int len) {
    write(b, 0, len);
}

void BufferedOutputStream::write(unsigned char* b, int off, int len) {
    /*
     * If the request length exceeds the size of the output buffer,
     * flush the output buffer and then write the data directly.
     * In this way buffered streams will cascade harmlessly.
     */
    if (len >= size) {
        flushBuffer();
        out->write(b, off, len);
        return;
    }
    if (len > size - count) {
        flushBuffer();
    }
    for (int i = 0; i < len; i++) {
        buf[count + i] = b[off + i];
    }
    count += len;
}

void BufferedOutputStream::flush() {
    flushBuffer();
    out->flush();
}

void BufferedOutputStream::close() {
    flush();
    out->close();
}

void BufferedOutputStream::flushBuffer() {
    if (count > 0) {
        out->write(buf, 0, count);
        count = 0;
    }
}

#endif /* __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_CPP__ */
```

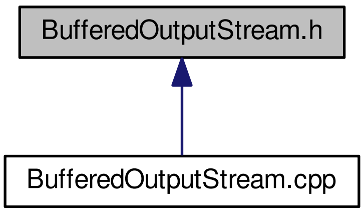
5.7 BufferedOutputStream.h File Reference

```
#include <FilterOutputStream.h>
```

Include dependency graph for BufferedOutputStream.h:



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



Classes

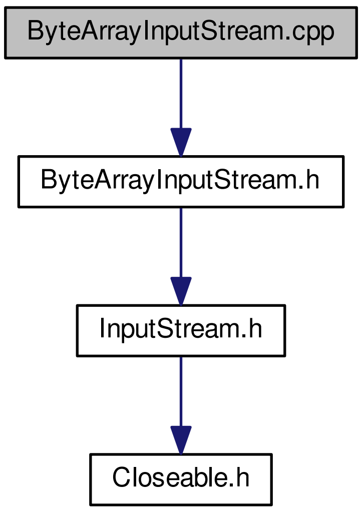
- class BufferedOutputStream

5.8 BufferedOutputStream.h

```
#ifndef __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_H__
#define __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_H__ 1

#include <FilterOutputStream.h>

class BufferedOutputStream : public FilterOutputStream {
protected:

    unsigned char* buf;

    int size;

    int count;

public:

    BufferedOutputStream(OutputStream* out, unsigned char* buf, int size
);

    void write(unsigned char b);

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char* b, int off, int len);

    virtual void flush();

    virtual void close();

private:

    void flushBuffer();
};

#endif /* __ARDUINO_IO_BUFFERED_OUTPUT_STREAM_H__ */
```

5.9 ByteArrayInputStream.cpp File Reference

`#include "ByteArrayInputStream.h"`
Include dependency graph for ByteArrayInputStream.cpp:



Macros

- #define __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_CPP__ 1

5.9.1 Macro Definition Documentation

5.9.1.1 #define __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_CPP__ 1

Arduino IO.

ByteArrayInputStream

A ByteArrayInputStream contains an internal buffer that contains bytes that may be read from the stream.

Definition at line 11 of file ByteArrayInputStream.cpp.

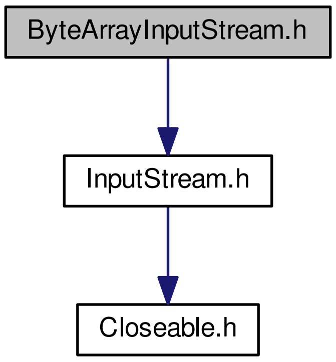
5.10 ByteArrayInputStream.cpp

```
00001
00010 #ifndef __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_CPP__ 1
00012
00013 #include "ByteArrayInputStream.h"
00014
00015 ByteArrayInputStream::ByteArrayInputStream(unsigned char* buf,
00016         unsigned int count) :
00017         buf(buf), count(count) {
00018     markpos = 0;
00019     pos = 0;
00020 }
00021
00022 int ByteArrayInputStream::available() {
00023     if ((count - pos) > 0) {
00024         return 1;
00025     }
00026     return 0;
00027 }
00028
00029 void ByteArrayInputStream::mark() {
00030     markpos = pos;
00031 }
00032
00033 bool ByteArrayInputStream::markSupported() {
00034     return true;
00035 }
00036
00037 int ByteArrayInputStream::read() {
00038     if (pos >= count) {
00039         return -1;
00040     }
00041     return buf[pos++];
00042 }
00043
00044 void ByteArrayInputStream::reset() {
00045     pos = markpos;
00046 }
00047
00048 #endif /* __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_CPP__ */
```

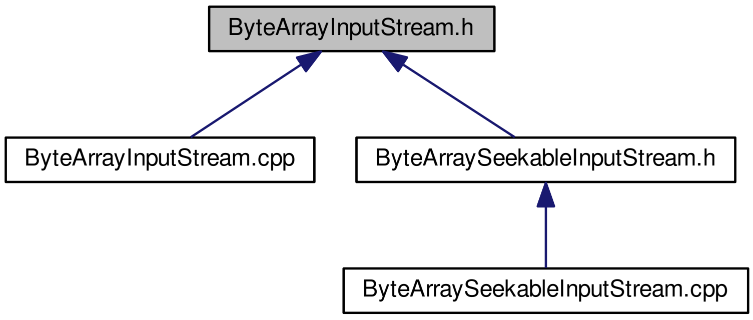
5.11 ByteArrayInputStream.h File Reference

`#include <InputStream.h>`

Include dependency graph for ByteArrayInputStream.h:



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


ByteArrayInputStream.h
ByteArrayInputStream.cpp
ByteArraySeekableInputStream.h
ByteArraySeekableInputStream.cpp

Classes

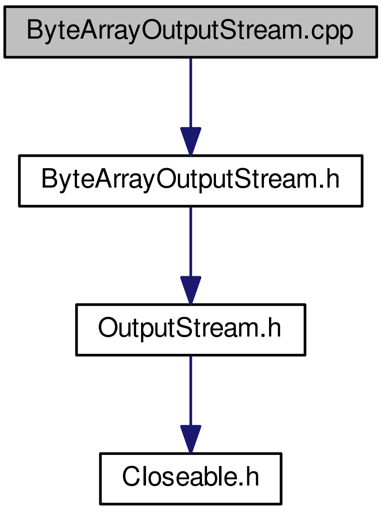
- class ByteArrayInputStream

5.12 ByteArrayInputStream.h

```
#ifndef __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_H__
#define __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_H__ 1

#include <InputStream.h>

class ByteArrayInputStream : public virtual InputStream {
protected:

    /*
     * The buffer where data is stored.
     */
    unsigned char* buf;

    /*
     * The number of valid bytes in the buffer.
     */
    unsigned int count;

    /*
     * Current position
     */
    unsigned int pos;

    /*
     * The currently marked position in the stream.
     */
    unsigned int markpos;

public:

    ByteArrayInputStream(unsigned char* buf, unsigned int count);

    virtual int available();

    virtual void mark();

    virtual bool markSupported();

    using InputStream::read;

    virtual int read();

    virtual void reset();
};

#endif /* __ARDUINO_IO_BYTE_ARRAY_INPUT_STREAM_H__ */
```

5.13 ByteArrayOutputStream.cpp File Reference

`#include "ByteArrayOutputStream.h"`
Include dependency graph for ByteArrayOutputStream.cpp:



Macros

- #define __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_CPP__ 1

5.13.1 Macro Definition Documentation

5.13.1.1 #define __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_CPP__ 1

Arduino IO.

ByteArrayOutputStream

This class implements an output stream in which the data is written into a unsigned char array.

Definition at line 11 of file ByteArrayOutputStream.cpp.

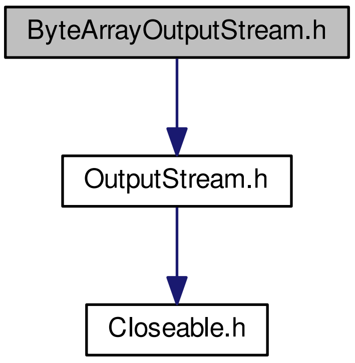
5.14 ByteArrayOutputStream.cpp

```
00001
00010 #ifndef __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_CPP__ 1
00012
00013 #include "ByteArrayOutputStream.h"
00014
00015 ByteArrayOutputStream::ByteArrayOutputStream(unsigned char* buf
      ,
00016         unsigned int count)
00017         : buf(buf), count(count) {
00018     pos = 0;
00019 }
00020
00021 void ByteArrayOutputStream::reset() {
00022     pos = 0;
00023 }
00024
00025 unsigned int ByteArrayOutputStream::size() {
00026     return count;
00027 }
00028
00029 unsigned char* ByteArrayOutputStream::toByteArray() {
00030     return buf;
00031 }
00032
00033 void ByteArrayOutputStream::write(unsigned char b) {
00034     buf[pos++] = b;
00035 }
00036
00037 #endif /* __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_CPP__ */
```

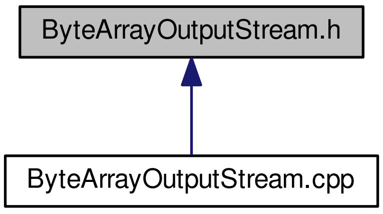
5.15 ByteArrayOutputStream.h File Reference

```
#include <OutputStream.h>
```

Include dependency graph for ByteArrayOutputStream.h:



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



Classes

- class ByteArrayOutputStream

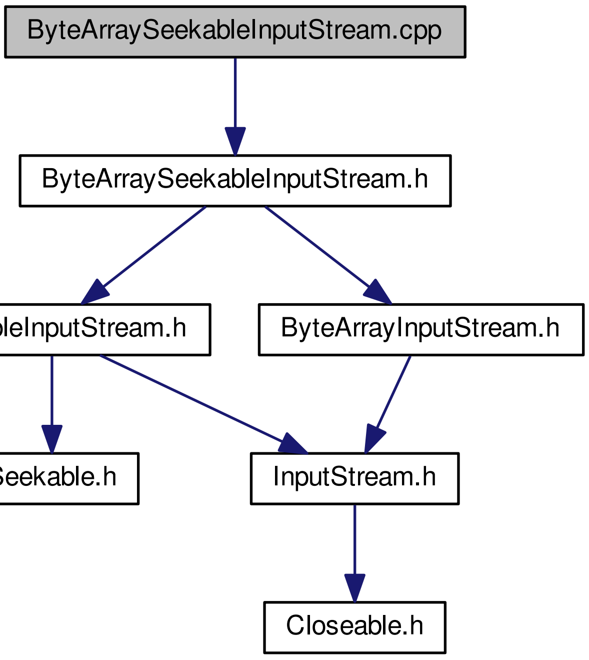
5.16 ByteArrayOutputStream.h

```

#ifndef __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_H__
#define __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_H__ 1

#include <OutputStream.h>

class ByteArrayOutputStream: public OutputStream {
protected:

    /*
     * The buffer where data is stored.
     */
    unsigned char* buf;

    /*
     * The number of valid bytes in the buffer.
     */
    unsigned int count;

    /*
     * Current position
     */
    unsigned int pos;

public:

    ByteArrayOutputStream(unsigned char* buf, unsigned int count);

    void reset();

    unsigned int size();

    unsigned char* toByteArray();

    using OutputStream::write;

    virtual void write(unsigned char b);
};

#endif /* __ARDUINO_IO_BYTE_ARRAY_OUTPUT_STREAM_H__ */
```

5.17 ByteArraySeekableInputStream.cpp File Reference

`#include "ByteArraySeekableInputStream.h"`

Include dependency graph for ByteArraySeekableInputStream.cpp:



Macros

- #define __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_CPP__ 1

5.17.1 Macro Definition Documentation

5.17.1.1 #define __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_CPP__ 1

Arduino IO.

ByteArraySeekableInputStream

A ByteArraySeekableInputStream obtains input bytes from a resource in a file system that implements Seekable←↩ InputStream interface.

Definition at line 11 of file ByteArraySeekableInputStream.cpp.

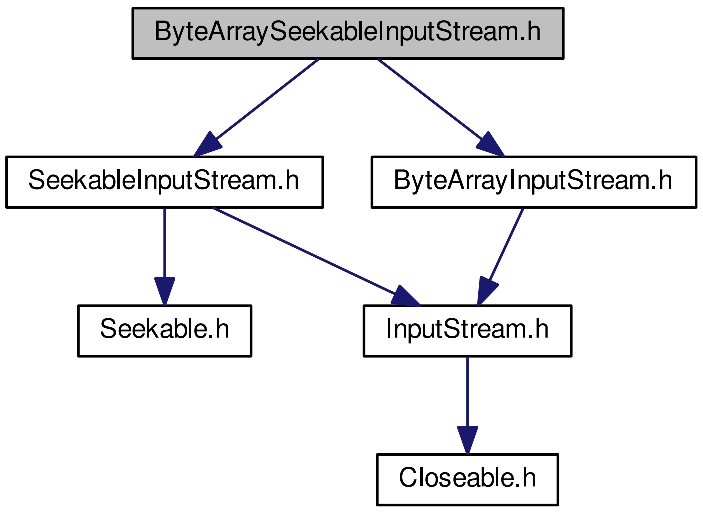
5.18 ByteArraySeekableInputStream.cpp

```
#ifndef __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_CPP__
#define __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_CPP__ 1

#include "ByteArraySeekableInputStream.h"

ByteArraySeekableInputStream::ByteArraySeekableInputStream
(unsigned char* buf,
        unsigned int count) :
        ByteArrayInputStream(buf, count) {
}

void ByteArraySeekableInputStream::seek(unsigned int pos) {
    this->pos = pos;
}

#endif /* __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_CPP__ */
```

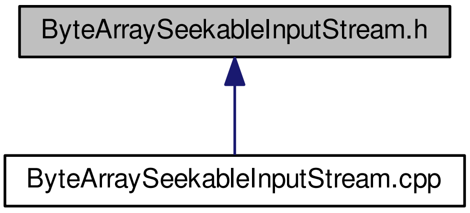
5.19 ByteArraySeekableInputStream.h File Reference

```
#include <SeekableInputStream.h>
#include <ByteArrayInputStream.h>
```

Include dependency graph for ByteArraySeekableInputStream.h:



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


ByteArraySeekableInputStream.h
ByteArraySeekableInputStream.cpp

Classes

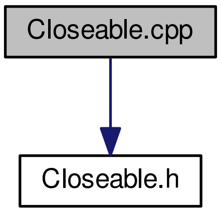
- class ByteArraySeekableInputStream

5.20 ByteArraySeekableInputStream.h

```
#ifndef __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_H__
#define __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_H__ 1

#include <SeekableInputStream.h>
#include <ByteArrayInputStream.h>

class ByteArraySeekableInputStream : public
SeekableInputStream,
        public ByteArrayInputStream {
public:

    ByteArraySeekableInputStream(unsigned char* buf, unsigned int
count);

    virtual void seek(unsigned int pos);
};

#endif /* __ARDUINO_IO_BYTE_ARRAY_SEEKABLE_INPUT_STREAM_H__ */
```

5.21 Closeable.cpp File Reference

`#include "Closeable.h"`
Include dependency graph for Closeable.cpp:



Macros

- #define __ARDUINO_IO_CLOSEABLE_CPP__ 1

5.21.1 Macro Definition Documentation

5.21.1.1 #define __ARDUINO_IO_CLOSEABLE_CPP__ 1

Arduino IO.

Closeable

A Closeable is a source or destination of data that can be closed.

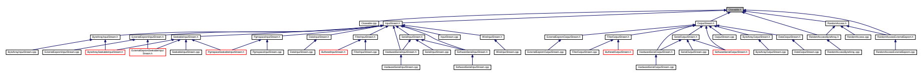
Definition at line 10 of file Closeable.cpp.

5.22 Closeable.cpp

```
00001
00009 #ifndef __ARDUINO_IO_CLOSEABLE_CPP__
00010 #define __ARDUINO_IO_CLOSEABLE_CPP__ 1
00011
00012 #include "Closeable.h"
00013
00014 #endif /* __ARDUINO_IO_CLOSEABLE_CPP__ */
```

5.23 Closeable.h File Reference

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



Classes

- class Closeable

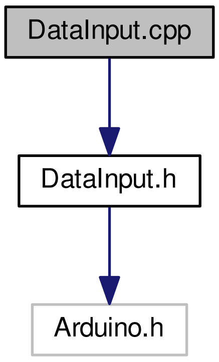
5.24 Closeable.h

```

#ifndef __ARDUINO_IO_CLOSEABLE_H__
#define __ARDUINO_IO_CLOSEABLE_H__ 1

class Closeable {
public:

    virtual void close() = 0;
};

#endif /* __ARDUINO_IO_CLOSEABLE_H__ */
```

5.25 DataInput.cpp File Reference

`#include "DataInput.h"`

Include dependency graph for DataInput.cpp:



Macros

- #define __ARDUINO_IO_DATA_INPUT_CPP__ 1

5.25.1 Macro Definition Documentation

5.25.1.1 #define __ARDUINO_IO_DATA_INPUT_CPP__ 1

Arduino IO.

DataInput

The DataInput interface provides for reading bytes from a binary stream and reconstructing from them data in any of the primitive types.

Definition at line 11 of file DataInput.cpp.

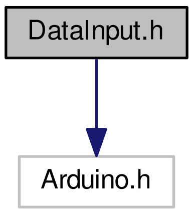
5.26 DataInput.cpp

```
00001
00010 #ifndef __ARDUINO_IO_DATA_INPUT_CPP__
00011 #define __ARDUINO_IO_DATA_INPUT_CPP__ 1
00012
00013 #include "DataInput.h"
00014
00015 #endif /* __ARDUINO_IO_DATA_INPUT_CPP__ */
00016
```

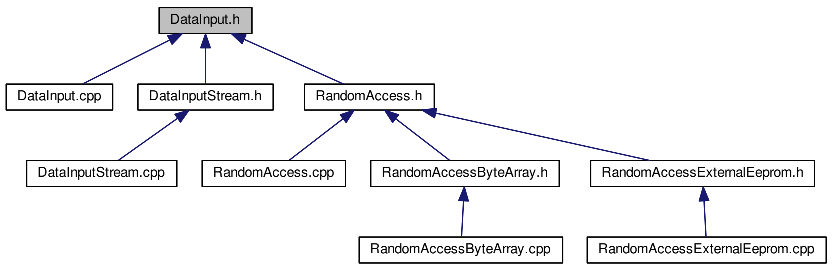
5.27 DataInput.h File Reference

`#include <Arduino.h>`

Include dependency graph for DataInput.h:



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



Classes

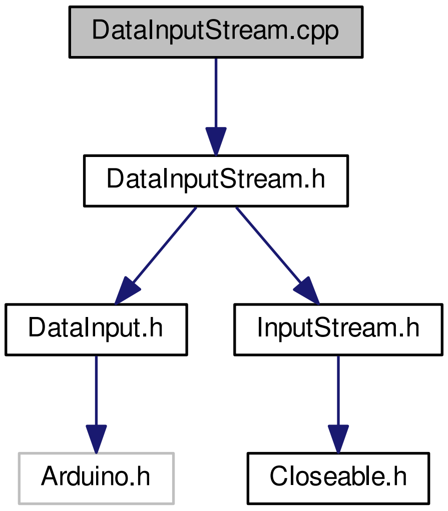
- class DataInput

5.28 DataInput.h

```
#ifndef __ARDUINO_IO_DATA_INPUT_H__
#define __ARDUINO_IO_DATA_INPUT_H__ 1

#include <Arduino.h>

class DataInput {
public:

    virtual unsigned char readByte() = 0;

    virtual bool readBoolean() = 0;

    virtual char readChar() = 0;

    virtual unsigned char readUnsignedChar() = 0;

    virtual int readInt() = 0;

    virtual unsigned int readUnsignedInt() = 0;

    virtual word readWord() = 0;

    virtual long readLong() = 0;

    virtual unsigned long readUnsignedLong() = 0;

    virtual float readFloat() = 0;

    virtual double readDouble() = 0;

    virtual void readFully(unsigned char* b, int len) = 0;

    virtual unsigned int skipBytes(unsigned int n) = 0;
};

#endif /* __ARDUINO_IO_DATA_INPUT_H__ */
```

5.29 DataInputStream.cpp File Reference

`#include "DataInputStream.h"`
Include dependency graph for DataInputStream.cpp:



Macros

- #define __ARDUINO_IO_DATA_INPUT_STREAM_CPP__ 1

5.29.1 Macro Definition Documentation

5.29.1.1 #define __ARDUINO_IO_DATA_INPUT_STREAM_CPP__ 1

Arduino IO.

DataInputStream

A data input stream lets an application read data from a InputStream.

Definition at line 10 of file DataInputStream.cpp.

5.30 DataInputStream.cpp

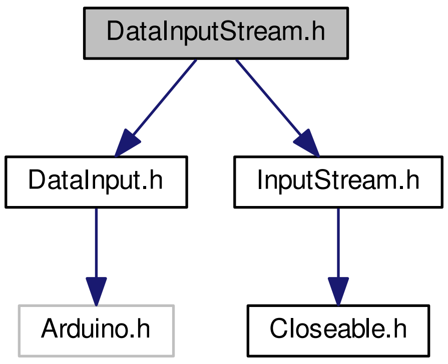
```

#ifndef __ARDUINO_IO_DATA_INPUT_STREAM_CPP__
```

```
#define __ARDUINO_IO_DATA_INPUT_STREAM_CPP__ 1

#include "DataInputStream.h"

DataInputStream::DataInputStream(InputStream* inputStream) :
        inputStream(inputStream) {
}

unsigned char DataInputStream::readByte() {
    return (unsigned char) inputStream->read();
}

bool DataInputStream::readBoolean() {
    return (bool) inputStream->read();
}

char DataInputStream::readChar() {
    return (char) inputStream->read();
}

unsigned char DataInputStream::readUnsignedChar() {
    return (unsigned char) inputStream->read();
}

int DataInputStream::readInt() {
    int v = 0;
    v = inputStream->read();
    v <<= 8;
    v |= (inputStream->read() & 0xff);
    return v;
}

unsigned int DataInputStream::readUnsignedInt() {
    return (unsigned int) readInt();
}

word DataInputStream::readWord() {
    return (word) readInt();
}

long DataInputStream::readLong() {
    long v = 0;
    v = inputStream->read();
    v <<= 8;
    v |= (inputStream->read() & 0xff);
    v <<= 8;
    v |= (inputStream->read() & 0xff);
    v <<= 8;
    v |= (inputStream->read() & 0xff);
    return v;
}

unsigned long DataInputStream::readUnsignedLong() {
    return (unsigned long) readLong();
}

float DataInputStream::readFloat() {
    return (float) readLong();
}

double DataInputStream::readDouble() {
    return (double) readLong();
}

void DataInputStream::readFully(unsigned char* b, int len) {
    for (int i = 0; i < len; i++) {
        b[i] = inputStream->read();
    }
}

unsigned int DataInputStream::skipBytes(unsigned int n) {
    return inputStream->skip(n);
}

#endif /* __ARDUINO_IO_DATA_INPUT_STREAM_CPP__ */
```

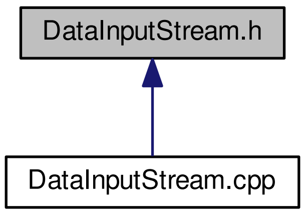
5.31 DataInputStream.h File Reference

```
#include <DataInput.h>
#include <InputStream.h>
```

Include dependency graph for DataInputStream.h:



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



Classes

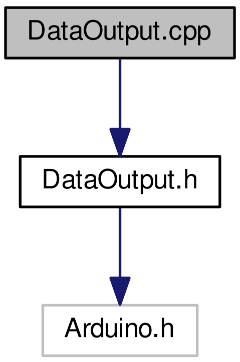
- class DataInputStream

5.32 DataInputStream.h

```

#ifndef __ARDUINO_IO_DATA_INPUT_STREAM_H__
#define __ARDUINO_IO_DATA_INPUT_STREAM_H__ 1

#include <DataInput.h>
#include <InputStream.h>

class DataInputStream : public DataInput {

    InputStream* inputStream;

public:

    DataInputStream(InputStream* inputStream);

    virtual unsigned char readByte();

    virtual bool readBoolean();

    virtual char readChar();

    virtual unsigned char readUnsignedChar();

    virtual int readInt();

    virtual unsigned int readUnsignedInt();

    virtual word readWord();

    virtual long readLong();

    virtual unsigned long readUnsignedLong();

    virtual float readFloat();

    virtual double readDouble();

    virtual void readFully(unsigned char* b, int len);

    virtual unsigned int skipBytes(unsigned int n);
};

#endif /* __ARDUINO_IO_DATA_INPUT_STREAM_H__ */
```

5.33 DataOutput.cpp File Reference

`#include "DataOutput.h"`
Include dependency graph for DataOutput.cpp:



Macros

- #define __ARDUINO_IO_DATA_OUTPUT_CPP__ 1

5.33.1 Macro Definition Documentation

5.33.1.1 #define __ARDUINO_IO_DATA_OUTPUT_CPP__ 1

Arduino IO.

DataOutput

The DataOutput interface provides for converting data from any of the primitive types to a series of bytes and writing these bytes to a binary stream.

Definition at line 11 of file DataOutput.cpp.

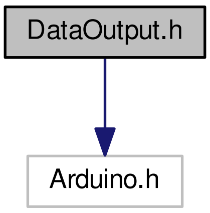
5.34 DataOutput.cpp

```
#ifndef __ARDUINO_IO_DATA_OUTPUT_CPP__
#define __ARDUINO_IO_DATA_OUTPUT_CPP__ 1

#include "DataOutput.h"

#endif /* __ARDUINO_IO_DATA_OUTPUT_CPP__ */

```

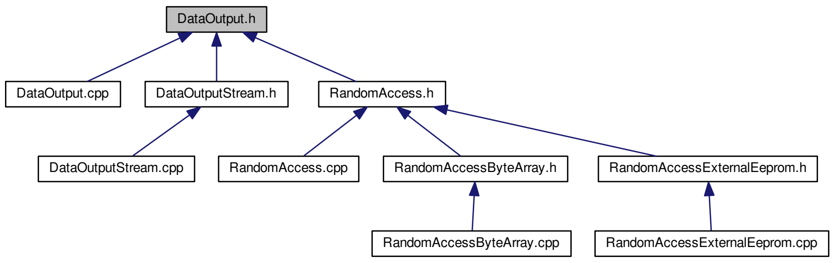
5.35 DataOutput.h File Reference

`#include <Arduino.h>`

Include dependency graph for DataOutput.h:



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



Classes

- class DataOutput

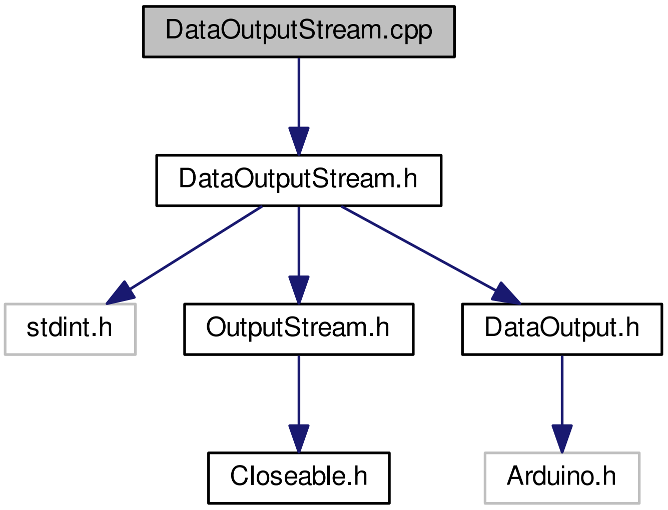
5.36 DataOutput.h

```
#ifndef __ARDUINO_IO_DATA_OUTPUT_H__
#define __ARDUINO_IO_DATA_OUTPUT_H__ 1

```

```
#include <Arduino.h>

class DataOutput {
public:

    virtual void write(unsigned char* b, int len) = 0;

    virtual void write(unsigned char b) = 0;

    virtual void writeByte(unsigned char b) = 0;

    virtual void writeBytes(unsigned char* b, int len) = 0;

    virtual void writeBoolean(bool v) = 0;

    virtual void writeChar(char c) = 0;

    virtual void writeUnsignedChar(unsigned char c) = 0;

    virtual void writeInt(int v) = 0;

    virtual void writeUnsignedInt(unsigned int v) = 0;

    virtual void writeWord(word v) = 0;

    virtual void writeLong(long v) = 0;

    virtual void writeUnsignedLong(unsigned long v) = 0;

    virtual void writeFloat(float v) = 0;

    virtual void writeDouble(double v) = 0;
};

#endif /* __ARDUINO_IO_DATA_OUTPUT_H__ */
```

5.37 DataOutputStream.cpp File Reference

`#include "DataOutputStream.h"`
Include dependency graph for DataOutputStream.cpp:



Macros

- #define __ARDUINO_IO_DATA_OUTPUT_STREAM_CPP__ 1

5.37.1 Macro Definition Documentation

5.37.1.1 #define __ARDUINO_IO_DATA_OUTPUT_STREAM_CPP__ 1

Arduino IO.

DataOutputStream

A data output stream lets an application write types to an OutputStream.

Definition at line 10 of file DataOutputStream.cpp.

5.38 DataOutputStream.cpp

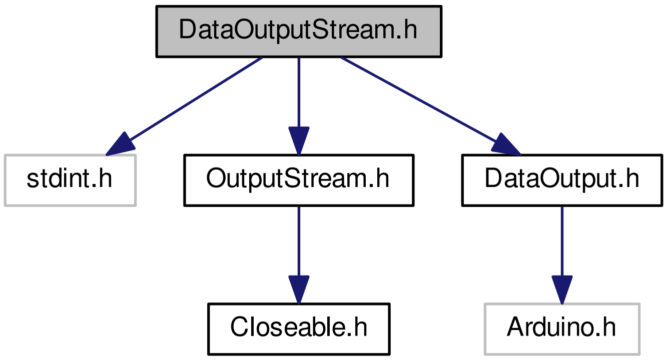
```
00001
00009 #ifndef __ARDUINO_IO_DATA_OUTPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_DATA_OUTPUT_STREAM_CPP__ 1
00011
00012 #include "DataOutputStream.h"
00013
00014 DataOutputStream::DataOutputStream(
      OutputStream* outputStream) :
00015         outputStream(outputStream) {
00016 }
00017
00018 void DataOutputStream::write(unsigned char* b, int len) {
00019     writeBytes(b, len);
00020 }
00021
00022 void DataOutputStream::write(unsigned char b) {
00023     writeByte(b);
00024 }
00025
00026 void DataOutputStream::writeByte(unsigned char b) {
00027     outputStream->write(b);
00028 }
00029
00030 void DataOutputStream::writeBytes(unsigned char* b, int len) {
00031     for (int i = 0; i < len; i++) {
00032         outputStream->write(b[i]);
00033     }
00034 }
00035
00036 void DataOutputStream::writeBoolean(bool v) {
00037     outputStream->write((unsigned char) v);
00038 }
00039
00040 void DataOutputStream::writeChar(char c) {
00041     outputStream->write((unsigned char) c);
00042 }
00043
00044 void DataOutputStream::writeUnsignedChar(unsigned char c) {
00045     outputStream->write((unsigned char) c);
00046 }
00047
00048 void DataOutputStream::writeInt(int v) {
00049     outputStream->write((unsigned char) ((v >> 8) & 0xff));
00050     outputStream->write((unsigned char) (v & 0xff));
00051 }
00052
00053 void DataOutputStream::writeUnsignedInt(unsigned int v) {
00054     writeInt((int) v);
00055 }
00056
00057 void DataOutputStream::writeWord(word v) {
00058     writeInt((int) v);
00059 }
00060
00061 void DataOutputStream::writeLong(long v) {
00062     outputStream->write((unsigned char) ((v >> 24) & 0xff));
00063     outputStream->write((unsigned char) ((v >> 16) & 0xff));
00064     outputStream->write((unsigned char) ((v >> 8) & 0xff));
00065     outputStream->write((unsigned char) (v & 0xff));
00066 }
00067
00068 void DataOutputStream::writeUnsignedLong(unsigned long v) {
00069     writeLong((long) v);
00070 }
00071
00072 void DataOutputStream::writeFloat(float v) {
00073     writeLong((long) v);
00074 }
```

```

void DataOutputStream::writeDouble(double v) {
    writeLong((long) v);
}

#endif /* __ARDUINO_IO_DATA_OUTPUT_STREAM_CPP__ */
```

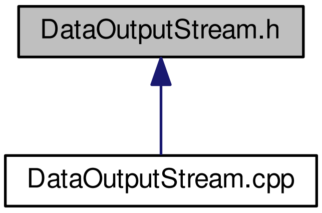
5.39 DataOutputStream.h File Reference

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

Include dependency graph for DataOutputStream.h:



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



Classes

- class DataOutputStream

5.40 DataOutputStream.h

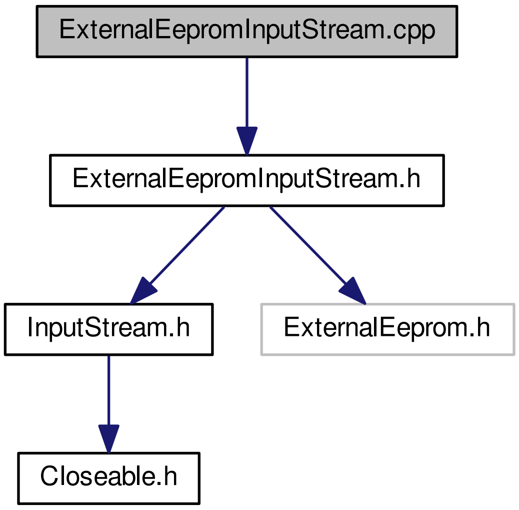
```
00001
```

```
#ifndef __ARDUINO_IO_DATA_OUTPUT_STREAM_H__
#define __ARDUINO_IO_DATA_OUTPUT_STREAM_H__ 1

#include <stdint.h>
#include <OutputStream.h>
#include <DataOutput.h>

class DataOutputStream : public DataOutput {

    OutputStream* outputStream;

public:

    DataOutputStream(OutputStream* outputStream);

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char b);

    virtual void writeByte(unsigned char b);

    virtual void writeBytes(unsigned char* b, int len);

    virtual void writeBoolean(bool v);

    virtual void writeChar(char c);

    virtual void writeUnsignedChar(unsigned char c);

    virtual void writeInt(int v);

    virtual void writeUnsignedInt(unsigned int v);

    virtual void writeWord(word v);

    virtual void writeLong(long v);

    virtual void writeUnsignedLong(unsigned long v);

    virtual void writeFloat(float v);

    virtual void writeDouble(double v);
};

#endif /* __ARDUINO_IO_DATA_OUTPUT_STREAM_H__ */
```

5.41 ExternalEepromInputStream.cpp File Reference

```
#include "ExternalEepromInputStream.h"
```

Include dependency graph for ExternalEepromInputStream.cpp:



Macros

- #define __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_CPP__ 1

5.41.1 Macro Definition Documentation

5.41.1.1 #define __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_CPP__ 1

Arduino IO.

ExternalEepromInputStream

An ExternalEepromInputStream obtains input bytes from a externalEeprom.

Definition at line 11 of file ExternalEepromInputStream.cpp.

5.42 ExternalEepromInputStream.cpp

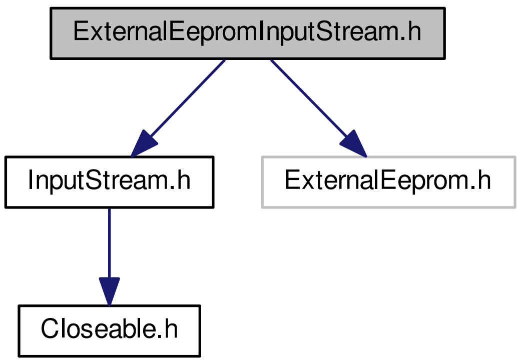
```
00001
00010 #ifndef __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_CPP__ 1
00012
00013 #include "ExternalEepromInputStream.h"
00014
00015 ExternalEepromInputStream::ExternalEepromInputStream(
00016         ExternalEeprom* externalEeprom) :
00017         externalEeprom(externalEeprom) {
00018     markpos = 0;
00019     pos = 0;
00020     externalEepromSize = externalEeprom->getDeviceSize();
00021 }
00022
00023 int ExternalEepromInputStream::available() {
00024     if (externalEepromSize > pos) {
00025         return 1;
00026     }
00027     return 0;
00028 }
```

```

void ExternalEepromInputStream::mark() {
    markpos = pos;
}

bool ExternalEepromInputStream::markSupported() {
    return true;
}

int ExternalEepromInputStream::read() {
    if (pos >= externalEepromSize) {
        return -1;
    }
    return (int) externalEeprom->read(pos++);
}

int ExternalEepromInputStream::read(unsigned char* b, int off, int len) {
    unsigned int available = (externalEepromSize -
pos);
    int cnt;
    len = (int) ((unsigned int) len > available) ? available : len;
    cnt = externalEeprom->readBytes(pos, &b[off], len);
    pos += cnt;
    return cnt;
}

void ExternalEepromInputStream::reset() {
    pos = markpos;
}

#endif /* __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_CPP__ */
```

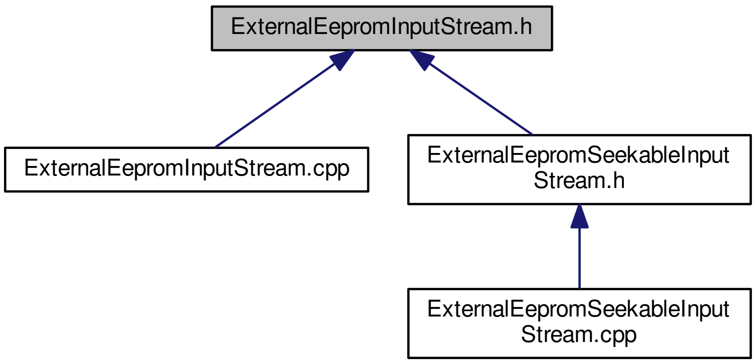
5.43 ExternalEepromInputStream.h File Reference

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

Include dependency graph for ExternalEepromInputStream.h:



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


ExternalEepromInputStream.h
ExternalEepromInputStream.cpp
ExternalEepromSeekableInputStream.h
ExternalEepromSeekableInputStream.cpp

Classes

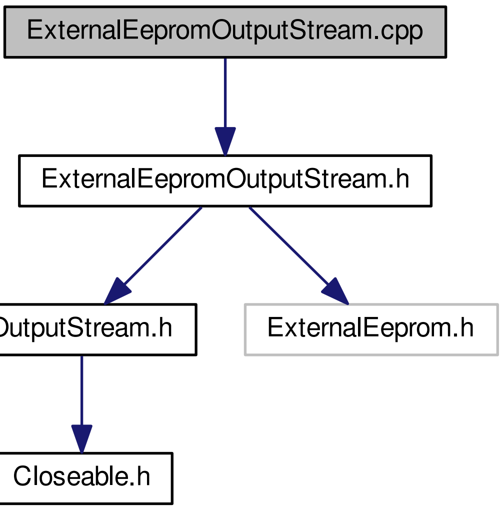
- class ExternalEepromInputStream

5.44 ExternalEepromInputStream.h

```
#ifndef __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_H__
#define __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_H__ 1

#include <InputStream.h>
#include <ExternalEeprom.h>

class ExternalEepromInputStream : public virtual
InputStream {
protected:

    /*
     * The externalEeprom where data is stored.
     */
    ExternalEeprom* externalEeprom;

    /*
     * Current position
     */
    unsigned int pos;

    /*
     * The currently marked position in the stream.
     */
    unsigned int markpos;

    /*
     * The size of the externalEeprom.
     */
    unsigned int externalEepromSize;

public:

    ExternalEepromInputStream(ExternalEeprom* externalEeprom);

    virtual int available();

    virtual void mark();

    virtual bool markSupported();

    using InputStream::read;

    virtual int read();

    virtual int read(unsigned char* b, int off, int len);

    virtual void reset();
};

#endif /* __ARDUINO_IO_EXTERNAL_EEPROM_INPUT_STREAM_H__ */
```

5.45 ExternalEepromOutputStream.cpp File Reference

`#include "ExternalEepromOutputStream.h"`
Include dependency graph for ExternalEepromOutputStream.cpp:



Macros

- #define __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_CPP__ 1

5.45.1 Macro Definition Documentation

5.45.1.1 #define __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_CPP__ 1

Arduino IO.

ExternalEepromOutputStream

A externalEeprom output stream is an output stream for writing data to a ExternalEeprom.

Definition at line 10 of file ExternalEepromOutputStream.cpp.

5.46 ExternalEepromOutputStream.cpp

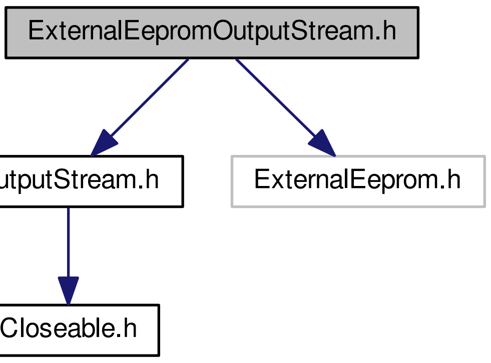
```
00001
00009 #ifndef __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_CPP__ 1
00011
00012 #include "ExternalEepromOutputStream.h"
00013
```

```
ExternalEepromOutputStream::ExternalEepromOutputStream
  (
        ExternalEeprom* externalEeprom) :
        externalEeprom(externalEeprom) {
    pos = 0;
}

void ExternalEepromOutputStream::write(unsigned char b) {
    externalEeprom->write(pos++, b);
}

void ExternalEepromOutputStream::write(unsigned char* b, int off, int len)
  {
    externalEeprom->writeBytes(pos, &b[off], len);
    pos += len;
}

#endif /* __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_CPP__ */
```

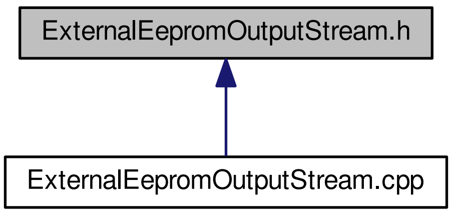
5.47 ExternalEepromOutputStream.h File Reference

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

Include dependency graph for ExternalEepromOutputStream.h:



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



Classes

- class ExternalEepromOutputStream

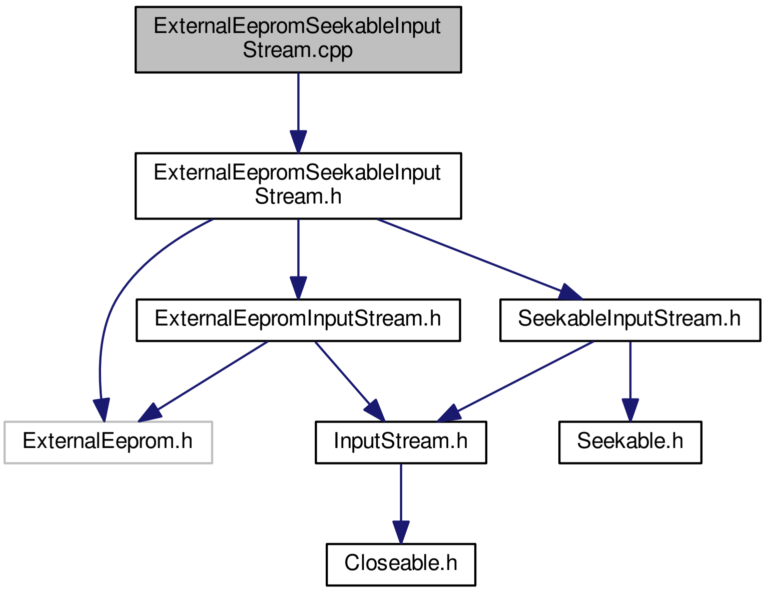
5.48 ExternalEepromOutputStream.h

```
#ifndef __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_H__
#define __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_H__ 1

#include <OutputStream.h>
#include <ExternalEeprom.h>

class ExternalEepromOutputStream : public OutputStream {

    ExternalEeprom* externalEeprom;

    unsigned int pos;

public:

    ExternalEepromOutputStream(ExternalEeprom* externalEeprom);

    using OutputStream::write;

    virtual void write(unsigned char b);

    virtual void write(unsigned char* b, int off, int len);
};

#endif /* __ARDUINO_IO_EXTERNAL_EEPROM_OUTPUT_STREAM_H__ */
```

5.49 ExternalEepromSeekableInputStream.cpp File Reference

`#include "ExternalEepromSeekableInputStream.h"`

Include dependency graph for ExternalEepromSeekableInputStream.cpp:


ExternalEepromSeekableInputStream.cpp
ExternalEepromSeekableInputStream.h
ExternalEepromInputStream.h
SeekableInputStream.h
ExternalEeprom.h
InputStream.h
Seekable.h
Closeable.h

Macros

- #define __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_CPP__ 1

5.49.1 Macro Definition Documentation

5.49.1.1 #define __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_CPP__ 1

Arduino IO.

ExternalEepromSeekableInputStream

A ExternalEepromSeekableInputStream obtains input bytes from a external input stream.

Definition at line 11 of file ExternalEepromSeekableInputStream.cpp.

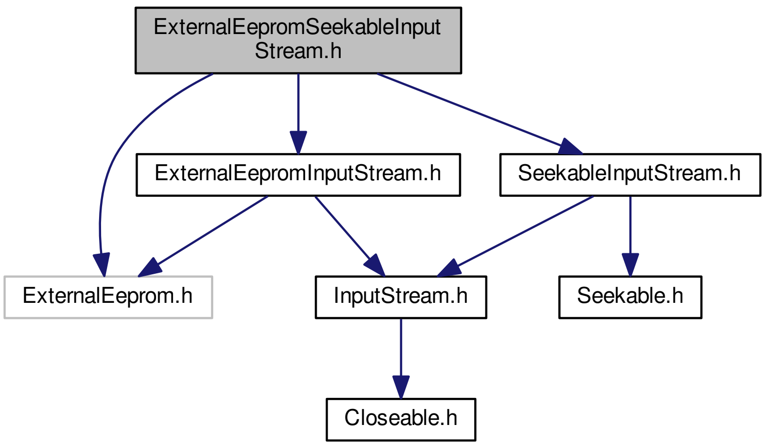
5.50 ExternalEepromSeekableInputStream.cpp

```
00001
00010 #ifndef __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_CPP__ 1
00012
00013 #include "ExternalEepromSeekableInputStream.h"
00014
00015 ExternalEepromSeekableInputStream::ExternalEepromSeekableInputStream
      (
00016         ExternalEeprom* externalEeprom) :
00017         ExternalEepromInputStream(externalEeprom) {
00018 }
00019
00020 void ExternalEepromSeekableInputStream::seek(unsigned int pos) {
00021     this->pos = pos;
00022 }
00023
00024 #endif /* __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_CPP__ */
```

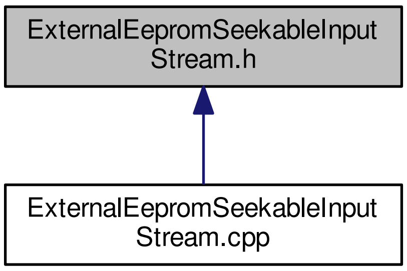
5.51 ExternalEepromSeekableInputStream.h File Reference

```
#include <ExternalEeprom.h>
#include <SeekableInputStream.h>
#include <ExternalEepromInputStream.h>
```

Include dependency graph for ExternalEepromSeekableInputStream.h:



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


ExternalEepromSeekableInput Stream.h
ExternalEepromSeekableInput Stream.cpp

Classes

- class ExternalEepromSeekableInputStream

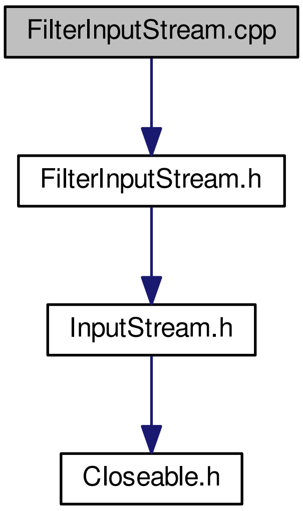
5.52 ExternalEepromSeekableInputStream.h

```

#ifndef __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_H__
#define __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_H__ 1

#include <ExternalEeprom.h>
#include <SeekableInputStream.h>
#include <ExternalEepromInputStream.h>

class ExternalEepromSeekableInputStream : public
ExternalEepromInputStream,
        public SeekableInputStream {
public:

    ExternalEepromSeekableInputStream(ExternalEeprom*
externalEeprom);

    virtual void seek(unsigned int pos);
};

#endif /* __ARDUINO_IO_EXTERNAL_EEPROM_SEEKABLE_INPUT_STREAM_H__ */
```

5.53 FilterInputStream.cpp File Reference

`#include "FilterInputStream.h"`

Include dependency graph for FilterInputStream.cpp:



Macros

- #define __ARDUINO_IO_FILTER_INPUT_STREAM_CPP__ 1

5.53.1 Macro Definition Documentation

5.53.1.1 #define __ARDUINO_IO_FILTER_INPUT_STREAM_CPP__ 1

A `FilterInputStream` contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.

The class `FilterInputStream` itself simply overrides all methods of `InputStream` with versions that pass all requests to the contained input stream. Subclasses of `FilterInputStream` may further override some of these methods and may also provide additional methods and fields.

Definition at line 17 of file FilterInputStream.cpp.

5.54 FilterInputStream.cpp

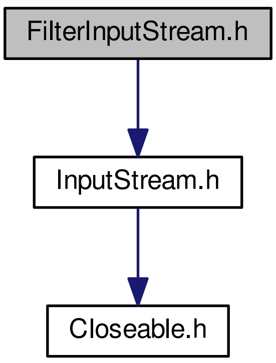
```
00001
00016 #ifndef __ARDUINO_IO_FILTER_INPUT_STREAM_CPP__
00017 #define __ARDUINO_IO_FILTER_INPUT_STREAM_CPP__ 1
00018
00019 #include "FilterInputStream.h"
00020
00021 FilterInputStream::FilterInputStream(
      InputStream* in) :
00022         in(in) {
00023 }
00024
00025 int FilterInputStream::read() {
00026     return in->read();
00027 }
00028
00029 int FilterInputStream::read(unsigned char* b, int len) {
00030     return in->read(b, len);
00031 }
```

```

int FilterInputStream::read(unsigned char* b, int off, int len) {
    return in->read(b, off, len);
}

unsigned int FilterInputStream::skip(unsigned int n) {
    return in->skip(n);
}

int FilterInputStream::available() {
    return in->available();
}

void FilterInputStream::close() {
    in->close();
}

void FilterInputStream::mark() {
    in->mark();
}

void FilterInputStream::reset() {
    in->reset();
}

bool FilterInputStream::markSupported() {
    return in->markSupported();
}

#endif /* __ARDUINO_IO_FILTER_INPUT_STREAM_CPP__ */
```

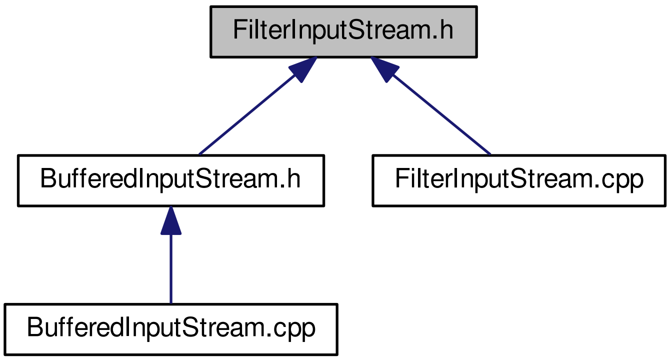
5.55 FilterInputStream.h File Reference

```
#include <InputStream.h>
```

Include dependency graph for FilterInputStream.h:



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



Classes

- class FilterInputStream

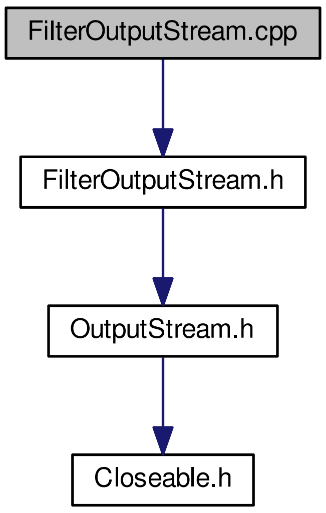
5.56 FilterInputStream.h

```
#ifndef __ARDUINO_IO_FILTER_INPUT_STREAM_H__
#define __ARDUINO_IO_FILTER_INPUT_STREAM_H__ 1

#include <InputStream.h>

class FilterInputStream : public virtual InputStream {

protected:

    InputStream* in;

    FilterInputStream(InputStream* in);

public:

    virtual int read();

    virtual int read(unsigned char* b, int len);

    virtual int read(unsigned char* b, int off, int len);

    virtual unsigned int skip(unsigned int n);

    virtual int available();

    virtual void close();

    virtual void mark();

    virtual void reset();

    virtual bool markSupported();
};

#endif /* __ARDUINO_IO_FILTER_INPUT_STREAM_H__ */
```

5.57 FilterOutputStream.cpp File Reference

```
#include "FilterOutputStream.h"
```

Include dependency graph for FilterOutputStream.cpp:



Macros

- #define __ARDUINO_IO_FILTER_OUTPUT_STREAM_CPP__ 1

5.57.1 Macro Definition Documentation

5.57.1.1 #define __ARDUINO_IO_FILTER_OUTPUT_STREAM_CPP__ 1

Arduino IO.

FilterOutputStream

This class is the superclass of all classes that filter output streams. These streams sit on top of an already existing output stream (the *underlying* output stream) which it uses as its basic sink of data, but possibly transforming the data along the way or providing additional functionality.

The class `FilterOutputStream` itself simply overrides all methods of `OutputStream` with versions that pass all requests to the underlying output stream. Subclasses of `FilterOutputStream` may further override some of these methods as well as provide additional methods and fields.

Definition at line 20 of file FilterOutputStream.cpp.

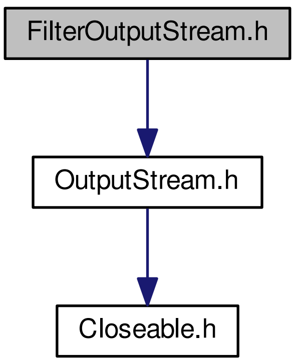
5.58 FilterOutputStream.cpp

```
00001
00019 #ifndef __ARDUINO_IO_FILTER_OUTPUT_STREAM_CPP__
00020 #define __ARDUINO_IO_FILTER_OUTPUT_STREAM_CPP__ 1
00021
00022 #include "FilterOutputStream.h"
00023
00024 FilterOutputStream::FilterOutputStream(
      OutputStream* out) :
00025         out(out) {
00026 }
00027
00028 void FilterOutputStream::write(unsigned char b) {
    out->write(b);
}

void FilterOutputStream::write(unsigned char* b, int len) {
    out->write(b, len);
}

void FilterOutputStream::write(unsigned char* b, int off, int len) {
    out->write(b, off, len);
}

void FilterOutputStream::flush() {
    out->flush();
}

void FilterOutputStream::close() {
    out->flush();
    out->close();
}

#endif /* __ARDUINO_IO_FILTER_OUTPUT_STREAM_CPP__ 1 */

```

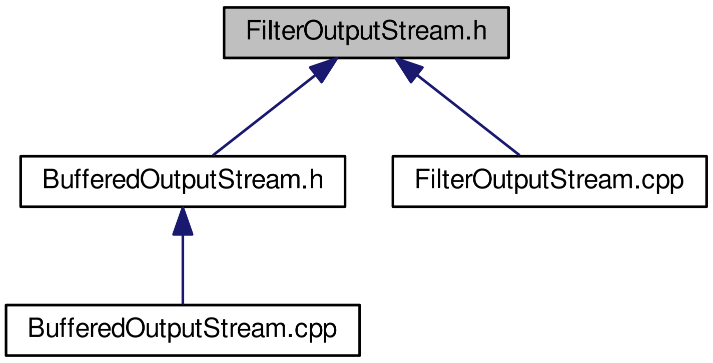
5.59 FilterOutputStream.h File Reference

`#include <OutputStream.h>`

Include dependency graph for FilterOutputStream.h:



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



Classes

- class FilterOutputStream

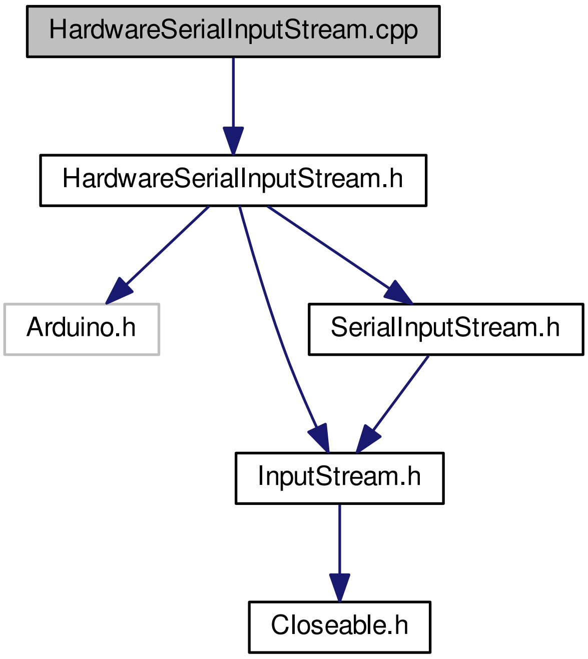
5.60 FilterOutputStream.h

```
#ifndef __ARDUINO_IO_FILTER_OUTPUT_STREAM_H__
#define __ARDUINO_IO_FILTER_OUTPUT_STREAM_H__ 1

#include <OutputStream.h>

class FilterOutputStream : public OutputStream {
protected:

    OutputStream* out;
public:

    FilterOutputStream(OutputStream* out);

    virtual void write(unsigned char b);

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char* b, int off, int len);

    virtual void flush();

    virtual void close();
};

#endif /* __ARDUINO_IO_FILTER_OUTPUT_STREAM_H__ */
```

5.61 HardwareSerialInputStream.cpp File Reference

```
#include "HardwareSerialInputStream.h"
```

Include dependency graph for HardwareSerialInputStream.cpp:



Macros

- #define __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_CPP__ 1

5.61.1 Macro Definition Documentation

5.61.1.1 #define __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_CPP__ 1

Arduino IO.

HardwareSerialInputStream

A HardwareSerialInputStream obtains input bytes from a serial port.

Definition at line 10 of file HardwareSerialInputStream.cpp.

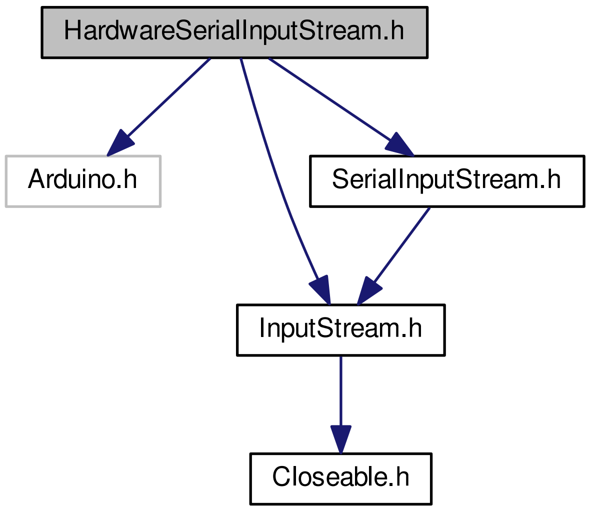
5.62 HardwareSerialInputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_CPP__ 1
00011
00012 #include "HardwareSerialInputStream.h"
00013
00014 HardwareSerialInputStream::HardwareSerialInputStream(
      unsigned int boudRate) {
00015     Serial.begin(boudRate);
00016 }
00017
00018 int HardwareSerialInputStream::available() {
00019     return Serial.available();
}

int HardwareSerialInputStream::read() {
    if (available() > 0) {
        return Serial.read();
    }
    return -1;
}

#endif /* __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_CPP__ */
```

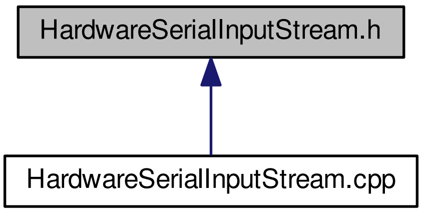
5.63 HardwareSerialInputStream.h File Reference

```
#include <Arduino.h>
#include <InputStream.h>
#include <SerialInputStream.h>
```

Include dependency graph for HardwareSerialInputStream.h:



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



Classes

- class HardwareSerialInputStream

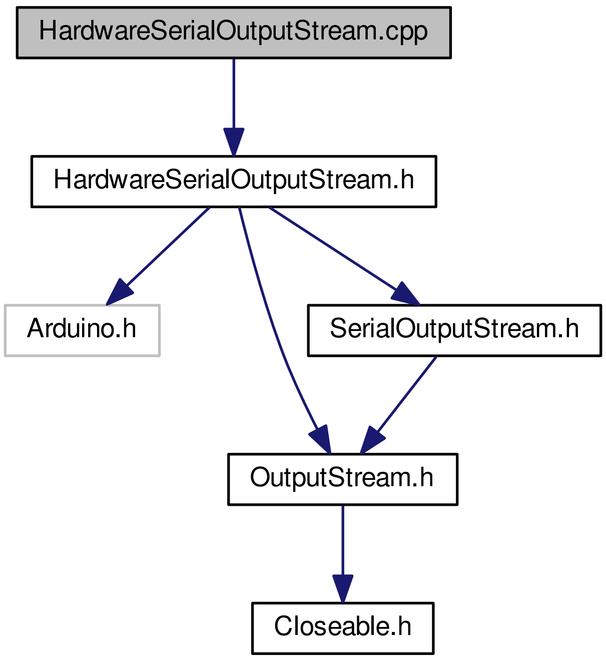
5.64 HardwareSerialInputStream.h

```
#ifndef __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_H__
#define __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_H__ 1

#include <Arduino.h>
#include <InputStream.h>
#include <SerialInputStream.h>

class HardwareSerialInputStream : public
SerialInputStream {
public:

    HardwareSerialInputStream(unsigned int boudRate);

    virtual int available();

    virtual int read();
};

#endif /* __ARDUINO_IO_HARDWARE_SERIAL_INPUT_STREAM_H__ */
```

5.65 HardwareSerialOutputStream.cpp File Reference

`#include "HardwareSerialOutputStream.h"`

Include dependency graph for HardwareSerialOutputStream.cpp:



Macros

- #define __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_CPP__ 1

5.65.1 Macro Definition Documentation

5.65.1.1 #define __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_CPP__ 1

Arduino IO.

HardwareSerialOutputStream

A software serial output stream is a output stream to write in a serial port.

Definition at line 10 of file HardwareSerialOutputStream.cpp.

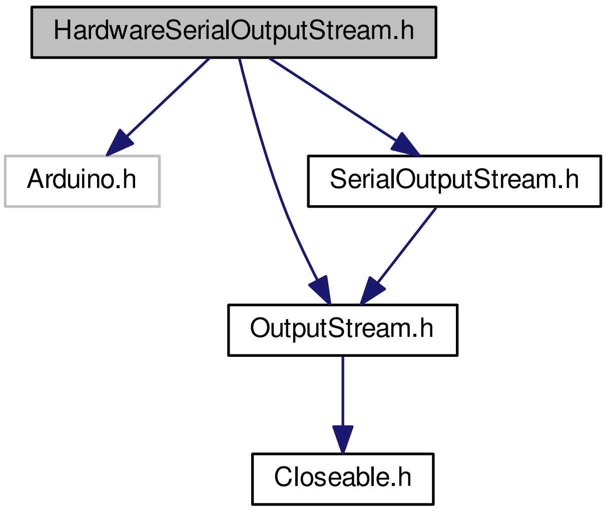
5.66 HardwareSerialOutputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_CPP__ 1
00011
00012 #include "HardwareSerialOutputStream.h"
00013
00014 HardwareSerialOutputStream::HardwareSerialOutputStream
      (unsigned int boudRate) {
00015     Serial.begin(9600);
00016 }
00017
00018 void HardwareSerialOutputStream::write(unsigned char b) {
00019     Serial.write(b);
00020 }
00021
00022 #endif /* __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_CPP__ */
```

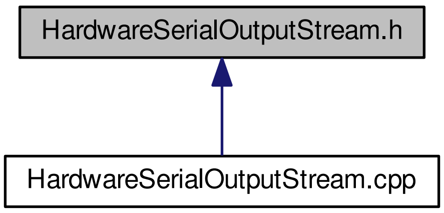
5.67 HardwareSerialOutputStream.h File Reference

```
#include <Arduino.h>
#include <OutputStream.h>
#include <SerialOutputStream.h>
```

Include dependency graph for HardwareSerialOutputStream.h:



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



Classes

- class HardwareSerialOutputStream

5.68 HardwareSerialOutputStream.h

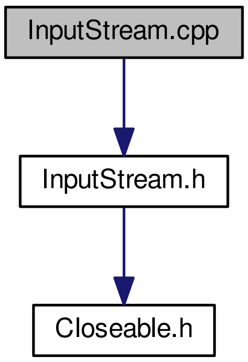
```
#ifndef __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_H__
#define __ARDUINO_IO_HARDWARE_SERIAL_OUTPUT_STREAM_H__ 1

#include <Arduino.h>
#include <OutputStream.h>
#include <SerialOutputStream.h>

class HardwareSerialOutputStream : public
      SerialOutputStream {
```

```
public:

    HardwareSerialOutputStream(unsigned int boudRate);

    virtual void write(unsigned char b);
};

#endif /* __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_H__ */
```

5.69 InputStream.cpp File Reference

`#include "InputStream.h"`

Include dependency graph for InputStream.cpp:



Macros

- #define __ARDUINO_IO_INPUT_STREAM_CPP__ 1

5.69.1 Macro Definition Documentation

5.69.1.1 #define __ARDUINO_IO_INPUT_STREAM_CPP__ 1

Arduino IO.

InputStream

This abstract class is the superclass of all classes representing an input stream of bytes.

Applications that need to define a subclass of InputStream must always provide a method that returns the next unsigned char of input.

Definition at line 14 of file InputStream.cpp.

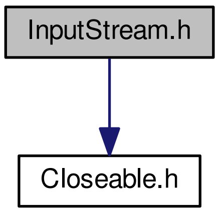
5.70 InputStream.cpp

```
00001
00013 #ifndef __ARDUINO_IO_INPUT_STREAM_CPP__
00014 #define __ARDUINO_IO_INPUT_STREAM_CPP__ 1
00015
00016 #include "InputStream.h"
00017
00018 int InputStream::available() {
    return 0;
}

void InputStream::close() {
}

void InputStream::mark() {
}

bool InputStream::markSupported() {
    return false;
}

int InputStream::read(unsigned char* b, int len) {
    return read(b, 0, len);
}

int InputStream::read(unsigned char* b, int off, int len) {
    int i, c;
    if (b == (unsigned char*) 0) {
        return 0;
    }
    c = read();
    if (c == -1) {
        return -1;
    }
    b[off] = (unsigned char) c;
    for (i = 1; i < len; i++) {
        c = read();
        if (c == -1) {
            break;
        }
        b[off + i] = (unsigned char) c;
    }
    return i;
}

void InputStream::reset() {
}

unsigned int InputStream::skip(unsigned int n) {
    unsigned int i;
    for (i = 0; i < n && available() > 0; i++) {
        read();
    }
    return i;
}

#endif /* __ARDUINO_IO_INPUT_STREAM_CPP__ */
```

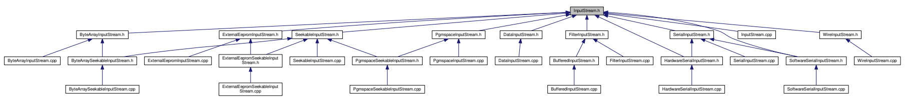
5.71 InputStream.h File Reference

`#include <Closeable.h>`

Include dependency graph for InputStream.h:



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


InputStream.h
ByteArrayInputStream.h
ExternalEepromInputStream.h
SeekableInputStream.h
PgmspaceInputStream.h
DataInputStream.h
FilterInputStream.h
SerialInputStream.h
InputStream.cpp
WireInputStream.h
ByteArrayInputStream.cpp
ByteArraySeekableInputStream.h
ExternalEepromInputStream.cpp
ExternalEepromSeekableInput Stream.h
SeekableInputStream.cpp
PgmspaceSeekableInputStream.h
PgmspaceInputStream.cpp
DataInputStream.cpp
BufferedInputStream.h
FilterInputStream.cpp
HardwareSerialInputStream.h
SerialInputStream.cpp
SoftwareSerialInputStream.h
WireInputStream.cpp
ByteArraySeekableInputStream.cpp
ExternalEepromSeekableInput Stream.cpp
PgmspaceSeekableInputStream.cpp
BufferedInputStream.cpp
HardwareSerialInputStream.cpp
SoftwareSerialInputStream.cpp

Classes

- class InputStream

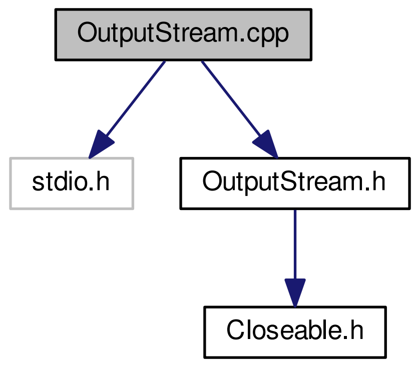
5.72 InputStream.h

```
#ifndef __ARDUINO_IO_INPUT_STREAM_H__
#define __ARDUINO_IO_INPUT_STREAM_H__ 1

#include <Closeable.h>

class InputStream : public Closeable {
public:

    virtual int available();

    virtual void close();

    virtual void mark();

    virtual bool markSupported();

    virtual int read() = 0;

    virtual int read(unsigned char* b, int len);

    virtual int read(unsigned char* b, int off, int len);

    virtual void reset();

    virtual unsigned int skip(unsigned int n);
};

#endif /* __ARDUINO_IO_INPUT_STREAM_H__ */
```

5.73 OutputStream.cpp File Reference

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

Include dependency graph for OutputStream.cpp:



Macros

- #define __ARDUINO_IO_OUTPUT_STREAM_CPP__ 1

5.73.1 Macro Definition Documentation

5.73.1.1 #define __ARDUINO_IO_OUTPUT_STREAM_CPP__ 1

Arduino IO.

OutputStream

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Definition at line 12 of file OutputStream.cpp.

5.74 OutputStream.cpp

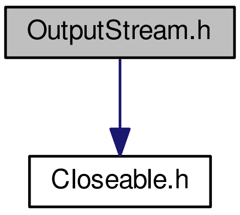
```
00001
00011 #ifndef __ARDUINO_IO_OUTPUT_STREAM_CPP__
00012 #define __ARDUINO_IO_OUTPUT_STREAM_CPP__ 1
00013
00014 #include <stdio.h>
00015
00016 #include "OutputStream.h"
00017
00018 void OutputStream::write(unsigned char* b, int len) {
00019     write(b, 0, len);
00020 }
00021
00022 void OutputStream::write(unsigned char* b, int off, int len) {
00023     if (b == (unsigned char*) 0 || len == 0) {
00024         return;
00025     }
00026     for (int i = 0; i < len; i++) {
00027         write(b[off + i]);
00028     }
00029 }
00030
00031 void OutputStream::flush() {
00032 }
00033
00034 void OutputStream::close() {
00035 }
```

```

#endif /* __ARDUINO_IO_OUTPUT_STREAM_CPP__ */
```

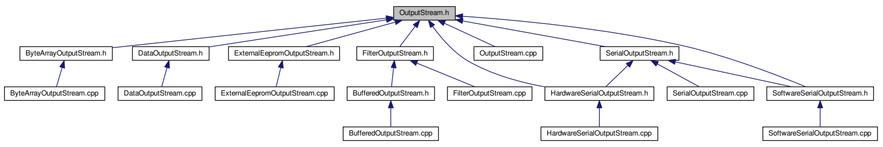
5.75 OutputStream.h File Reference

`#include <Closeable.h>`

Include dependency graph for OutputStream.h:



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



Classes

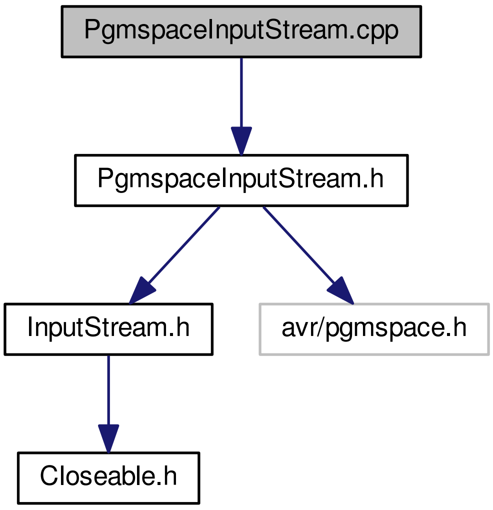
- class OutputStream

5.76 OutputStream.h

```

#ifndef __ARDUINO_IO_OUTPUT_STREAM_H__
#define __ARDUINO_IO_OUTPUT_STREAM_H__ 1

#include <Closeable.h>

class OutputStream : public Closeable {
public:

    virtual void close();

    virtual void flush();

    virtual void write(unsigned char b) = 0;

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char* b, int off, int len);
};

#endif /* __ARDUINO_IO_OUTPUT_STREAM_H__ */
```

5.77 PgmspaceInputStream.cpp File Reference

`#include "PgmspaceInputStream.h"`
Include dependency graph for PgmspaceInputStream.cpp:



Macros

- #define __ARDUINO_IO_PGMSPACE_INPUT_STREAM_CPP__ 1

5.77.1 Macro Definition Documentation

5.77.1.1 #define __ARDUINO_IO_PGMSPACE_INPUT_STREAM_CPP__ 1

Arduino IO.

PgmspaceInputStream

A PgmspaceInputStream contains an internal buffer that contains bytes that may be read from the stream mapped to an part of the pgmspace.

Definition at line 11 of file PgmspaceInputStream.cpp.

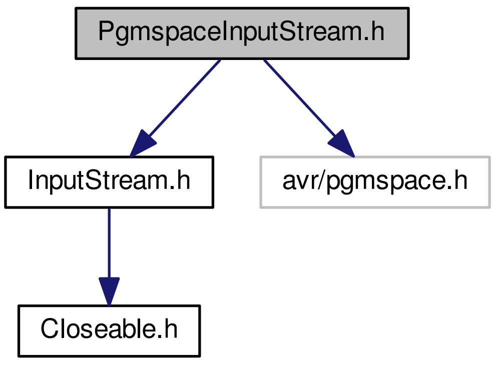
5.78 PgmspaceInputStream.cpp

```
00001
00010 #ifndef __ARDUINO_IO_PGMSPACE_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_PGMSPACE_INPUT_STREAM_CPP__ 1
00012
00013 #include "PgmspaceInputStream.h"
00014
00015 PgmspaceInputStream::PgmspaceInputStream(char PROGMEM* buf,
      unsigned int count) : buf(buf), count(count) {
00016     markpos = 0;
00017     pos = 0;
00018 }
00019
00020 int PgmspaceInputStream::available() {
00021     if ((count - pos) > 0) {
        return 1;
    }
    return 0;
}

void PgmspaceInputStream::mark() {
    markpos = pos;
}

bool PgmspaceInputStream::markSupported() {
    return true;
}

int PgmspaceInputStream::read() {
    return pgm_read_byte(buf + pos++);
}

void PgmspaceInputStream::reset() {
    pos = markpos;
}

#endif /* __ARDUINO_IO_PGMSPACE_INPUT_STREAM_CPP__ */
```

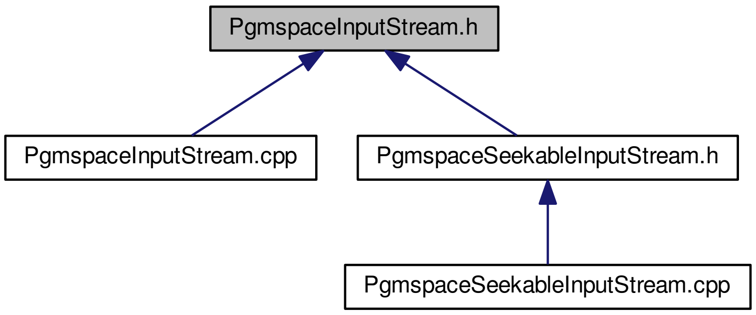
5.79 PgmspaceInputStream.h File Reference

```
#include <InputStream.h>
#include <avr/pgmspace.h>
```

Include dependency graph for PgmspaceInputStream.h:



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


PgmspaceInputStream.h
PgmspaceInputStream.cpp
PgmspaceSeekableInputStream.h
PgmspaceSeekableInputStream.cpp

Classes

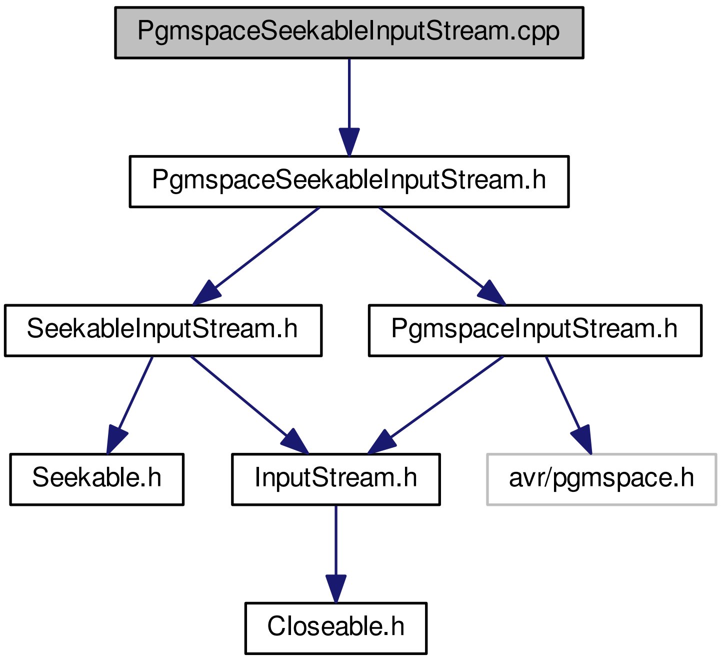
- class PgmspaceInputStream

5.80 PgmspaceInputStream.h

```

#ifndef __ARDUINO_IO_PGMSPACE_INPUT_STREAM_H__
#define __ARDUINO_IO_PGMSPACE_INPUT_STREAM_H__ 1

#include <InputStream.h>
#include <avr/pgmspace.h>

class PgmspaceInputStream : public virtual InputStream {
protected:

    /*
     * The buffer where data is stored.
     */
    char PROGMEM* buf;

    /*
     * The number of valid bytes in the buffer.
     */
    unsigned int count;

    /*
     * Current position
     */
    unsigned int pos;

    /*
     * The currently marked position in the stream.
     */
    unsigned int markpos;

public:

    explicit PgmspaceInputStream(char PROGMEM* buf, unsigned int count);

    virtual int available();

    virtual void mark();

    virtual bool markSupported();

    using InputStream::read;

    virtual int read();

    virtual void reset();
};

#endif /* __ARDUINO_IO_PGMSPACE_INPUT_STREAM_H__ */
```

5.81 PgmspaceSeekableInputStream.cpp File Reference

`#include "PgmspaceSeekableInputStream.h"`
Include dependency graph for PgmspaceSeekableInputStream.cpp:



Macros

- #define __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_CPP__ 1

5.81.1 Macro Definition Documentation

5.81.1.1 #define __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_CPP__ 1

Arduino IO.

PgmspaceSeekableInputStream

A PgmspaceSeekableInputStream obtains input bytes from a resource in a file system that implements Seekable↩
InputStream interface.

Definition at line 11 of file PgmspaceSeekableInputStream.cpp.

5.82 PgmspaceSeekableInputStream.cpp

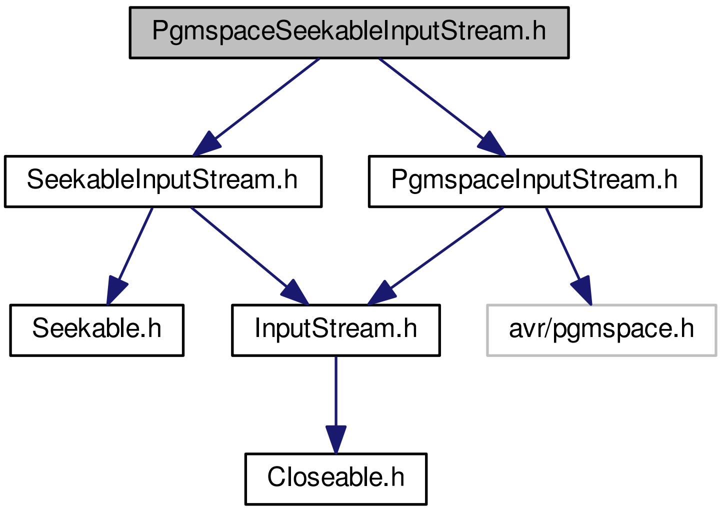
```
00001
00010 #ifndef __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_CPP__ 1
00012
00013 #include "PgmspaceSeekableInputStream.h"
00014
00015 PgmspaceSeekableInputStream::PgmspaceSeekableInputStream
```

```
      (char PROGMEM* buf, unsigned int count) : PgmspaceInputStream(buf, count) {
}

void PgmspaceSeekableInputStream::seek(unsigned int pos) {
    this->pos = pos;
}

#endif /* __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_CPP__ */
```

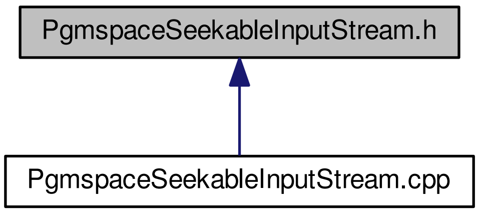
5.83 PgmspaceSeekableInputStream.h File Reference

```
#include <SeekableInputStream.h>
#include <PgmspaceInputStream.h>
```

Include dependency graph for PgmspaceSeekableInputStream.h:



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


PgmspaceSeekableInputStream.h
PgmspaceSeekableInputStream.cpp

Classes

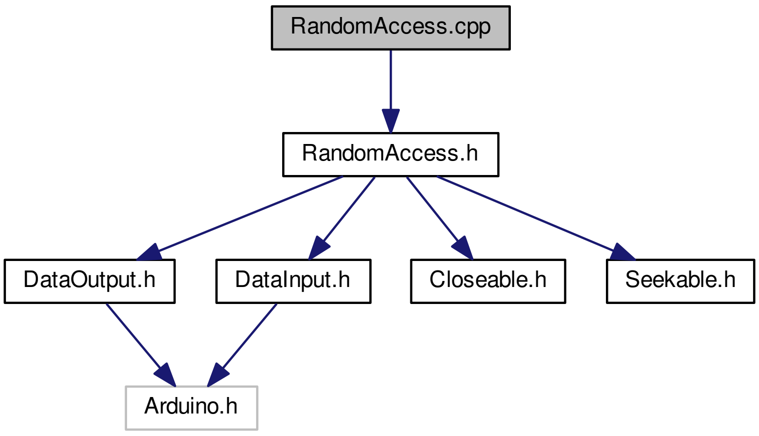
- class PgmspaceSeekableInputStream

5.84 PgmspaceSeekableInputStream.h

```
#ifndef __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_H__
#define __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_H__ 1

#include <SeekableInputStream.h>
#include <PgmspaceInputStream.h>

class PgmspaceSeekableInputStream : public
SeekableInputStream,
        public PgmspaceInputStream {
public:

    PgmspaceSeekableInputStream(char PROGMEM* buf, unsigned int
count);

    virtual void seek(unsigned int pos);
};

#endif /* __ARDUINO_IO_PGMSPACE_SEEKABLE_INPUT_STREAM_H__ */
```

5.85 RandomAccess.cpp File Reference

`#include "RandomAccess.h"`
Include dependency graph for RandomAccess.cpp:



Macros

- #define __ARDUINO_IO_RANDOM_ACCESS_CPP__ 1

5.85.1 Macro Definition Documentation

5.85.1.1 #define __ARDUINO_IO_RANDOM_ACCESS_CPP__ 1

Araduino IO.

RandomAccess

Interface derived from DataInput, DataOutput, Closeable and Seekable.

Definition at line 10 of file RandomAccess.cpp.

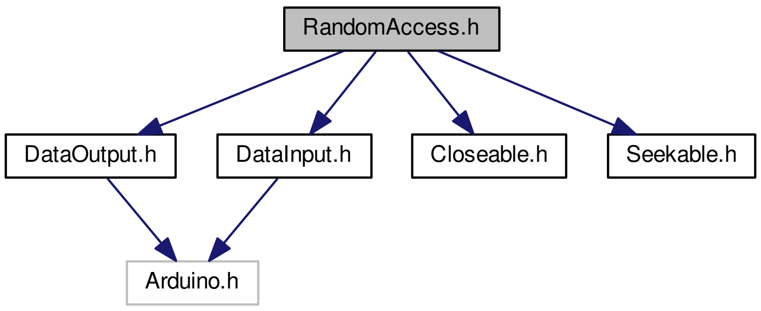
5.86 RandomAccess.cpp

```

#ifndef __ARDUINO_IO_RANDOM_ACCESS_CPP__
#define __ARDUINO_IO_RANDOM_ACCESS_CPP__ 1

#include "RandomAccess.h"

#endif /* __ARDUINO_IO_RANDOM_ACCESS_CPP__ */
```

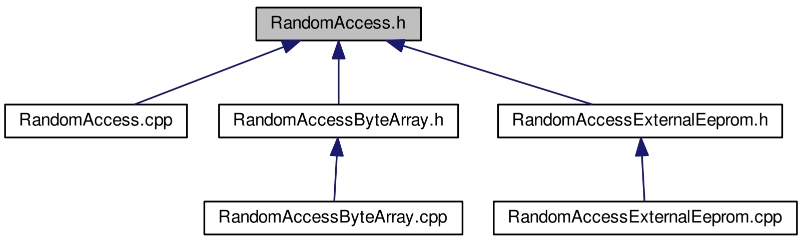
5.87 RandomAccess.h File Reference

```
#include <DataOutput.h>
#include <DataInput.h>
#include <Closeable.h>
#include <Seekable.h>
```

Include dependency graph for RandomAccess.h:



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


RandomAccess.h
RandomAccess.cpp
RandomAccessByteArray.h
RandomAccessExternalEeprom.h
RandomAccessByteArray.cpp
RandomAccessExternalEeprom.cpp

Classes

- class RandomAccess

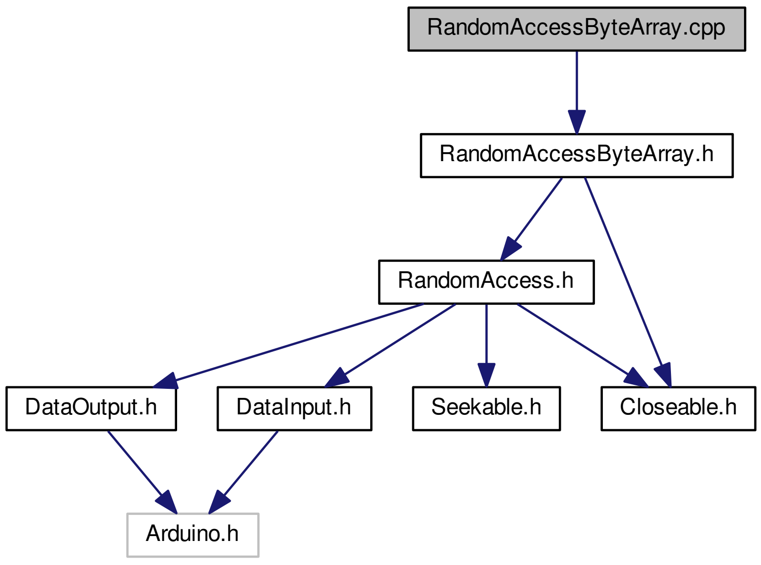
5.88 RandomAccess.h

```

#ifndef __ARDUINO_IO_RANDOM_ACCESS_H__
```

```
#define __ARDUINO_IO_RANDOM_ACCESS_H__ 1

#include <DataOutput.h>
#include <DataInput.h>
#include <Closeable.h>
#include <Seekable.h>

class RandomAccess : public virtual DataOutput, public virtual
     DataInput,
        public virtual Closeable, public virtual Seekable {
};

#endif /* __ARDUINO_IO_RANDOM_ACCESS_H__ */
```

5.89 RandomAccessByteArray.cpp File Reference

`#include "RandomAccessByteArray.h"`
Include dependency graph for RandomAccessByteArray.cpp:



Macros

- #define __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_CPP__ 1

5.89.1 Macro Definition Documentation

5.89.1.1 #define __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_CPP__ 1

Araduino IO.

RandomAccessByteArray

Instances of this class support both reading and writing to a random access unsigned char array.

Definition at line 11 of file RandomAccessByteArray.cpp.

5.90 RandomAccessByteArray.cpp

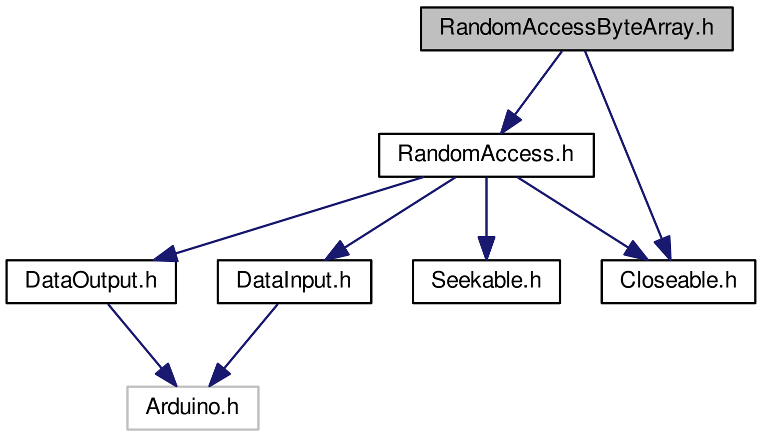
```
#ifndef __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_CPP__
#define __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_CPP__ 1

#include "RandomAccessByteArray.h"

RandomAccessByteArray::RandomAccessByteArray(unsigned char* buf
,
        unsigned int count) :
        buf(buf), count(count) {
    pos = 0;
}

unsigned int RandomAccessByteArray::length() {
    return count;
}

void RandomAccessByteArray::seek(unsigned int pos) {
    this->pos = pos;
}

void RandomAccessByteArray::close() {
}

void RandomAccessByteArray::write(unsigned char* b, int len) {
    writeBytes(b, len);
}

void RandomAccessByteArray::write(unsigned char b) {
    buf[pos++] = b;
}

void RandomAccessByteArray::writeByte(unsigned char b) {
    buf[pos++] = b;
}

void RandomAccessByteArray::writeBytes(unsigned char* b, int len) {
    for (int i = 0; i < len; i++) {
        buf[pos++] = b[i];
    }
}

void RandomAccessByteArray::writeBoolean(bool v) {
    buf[pos++] = (unsigned char) v;
}

void RandomAccessByteArray::writeChar(char c) {
    buf[pos++] = (unsigned char) c;
}

void RandomAccessByteArray::writeUnsignedChar(unsigned char c) {
    buf[pos++] = (unsigned char) c;
}

void RandomAccessByteArray::writeInt(int v) {
    buf[pos++] = (unsigned char) ((v >> 8) & 0xff);
    buf[pos++] = (unsigned char) (v & 0xff);
}

void RandomAccessByteArray::writeUnsignedInt(unsigned int v) {
    writeInt((int) v);
}

void RandomAccessByteArray::writeWord(word v) {
    writeInt((int) v);
}

void RandomAccessByteArray::writeLong(long v) {
    buf[pos++] = (unsigned char) ((v >> 24) & 0xff);
    buf[pos++] = (unsigned char) ((v >> 16) & 0xff);
    buf[pos++] = (unsigned char) ((v >> 8) & 0xff);
    buf[pos++] = (unsigned char) (v & 0xff);
}

void RandomAccessByteArray::writeUnsignedLong(unsigned long v) {
    writeLong((long) v);
}

void RandomAccessByteArray::writeFloat(float v) {
    writeLong((long) v);
}

void RandomAccessByteArray::writeDouble(double v) {
    writeLong((long) v);
}
```

```

unsigned char RandomAccessByteArray::readByte() {
    return buf[pos++];
}

bool RandomAccessByteArray::readBoolean() {
    return (bool) buf[pos++];
}

char RandomAccessByteArray::readChar() {
    return (char) buf[pos++];
}

unsigned char RandomAccessByteArray::readUnsignedChar() {
    return (unsigned char) buf[pos++];
}

int RandomAccessByteArray::readInt() {
    int v = 0;
    v = buf[pos++];
    v <<= 8;
    v |= buf[pos++];
    return v;
}

unsigned int RandomAccessByteArray::readUnsignedInt() {
    return (unsigned int) readInt();
}

word RandomAccessByteArray::readWord() {
    return (word) readInt();
}

long RandomAccessByteArray::readLong() {
    long v = 0;
    v = (buf[pos++] & 0xff);
    v <<= 8;
    v |= (buf[pos++] & 0xff);
    v <<= 8;
    v |= (buf[pos++] & 0xff);
    v <<= 8;
    v |= (buf[pos++] & 0xff);
    return v;
}

unsigned long RandomAccessByteArray::readUnsignedLong() {
    return (unsigned long) readLong();
}

float RandomAccessByteArray::readFloat() {
    return (float) readLong();
}

double RandomAccessByteArray::readDouble() {
    return (double) readLong();
}

void RandomAccessByteArray::readFully(unsigned char* b, int len) {
    for (int i = 0; i < len; i++) {
        b[i] = buf[pos++];
    }
}

unsigned int RandomAccessByteArray::skipBytes(unsigned int n) {
    unsigned int skipped;
    unsigned int newpos;
    newpos = pos + n;
    if (newpos > count) {
        newpos = count;
    }
    skipped = newpos - pos;
    pos = newpos;
    return skipped;
}

#endif /* __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_CPP__ */
```

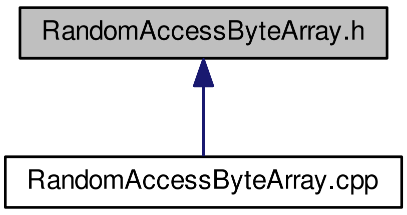
5.91 RandomAccessByteArray.h File Reference

```
#include <RandomAccess.h>
#include <Closeable.h>
```

Include dependency graph for RandomAccessByteArray.h:



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



Classes

- class RandomAccessByteArray

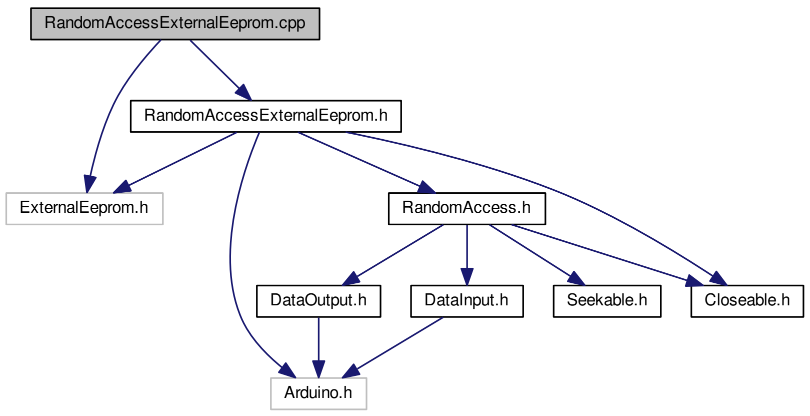
5.92 RandomAccessByteArray.h

```

#ifndef __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_H__
#define __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_H__ 1

#include <RandomAccess.h>
#include <Closeable.h>

class RandomAccessByteArray : public RandomAccess, public virtual
Closeable {

    unsigned char* buf;

    unsigned int count;

    unsigned int pos;

public:

    RandomAccessByteArray(unsigned char* buf, unsigned int count);

    virtual void seek(unsigned int pos);

    unsigned int length();

    virtual void close();

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char b);

    virtual void writeByte(unsigned char b);

    virtual void writeBytes(unsigned char* b, int len);

    virtual void writeBoolean(bool v);

    virtual void writeChar(char c);

    virtual void writeUnsignedChar(unsigned char c);

    virtual void writeInt(int v);

    virtual void writeUnsignedInt(unsigned int v);

    virtual void writeWord(word v);

    virtual void writeLong(long v);

    virtual void writeUnsignedLong(unsigned long v);

    virtual void writeFloat(float v);

    virtual void writeDouble(double v);

    virtual unsigned char readByte();

    virtual bool readBoolean();

    virtual char readChar();

    virtual unsigned char readUnsignedChar();

    virtual int readInt();

    virtual unsigned int readUnsignedInt();

    virtual word readWord();

    virtual long readLong();

    virtual unsigned long readUnsignedLong();

    virtual float readFloat();

    virtual double readDouble();

    virtual void readFully(unsigned char* b, int len);

    virtual unsigned int skipBytes(unsigned int n);
};
#endif /* __ARDUINO_IO_RANDOM_ACCESS_BYTE_ARRAY_H__ */
```

5.93 RandomAccessExternalEeprom.cpp File Reference

```
#include <ExternalEeprom.h>
#include "RandomAccessExternalEeprom.h"
```

Include dependency graph for RandomAccessExternalEeprom.cpp:



Macros

- #define __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_CPP__ 1

5.93.1 Macro Definition Documentation

5.93.1.1 #define __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_CPP__ 1

Araduino IO.

RandomAccessExternalEeprom

Instances of this class support both reading and writing to a random access externalEeprom. A random access externalEeprom behaves like a large array of bytes stored in the externalEeprom system.

Definition at line 12 of file RandomAccessExternalEeprom.cpp.

5.94 RandomAccessExternalEeprom.cpp

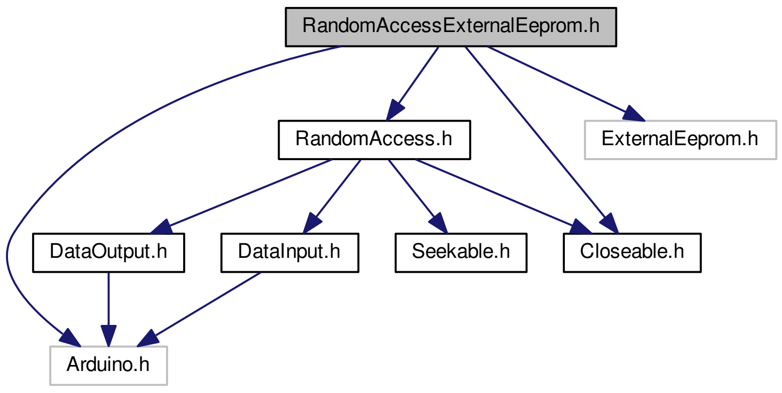
```
00001
00011 #ifndef __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_CPP__
00012 #define __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_CPP__ 1
00013
00014 #include <ExternalEeprom.h>
00015 #include "RandomAccessExternalEeprom.h"
00016
00017 RandomAccessExternalEeprom::RandomAccessExternalEeprom
      (
00018         ExternalEeprom* externalEeprom) :
00019         externalEeprom(externalEeprom) {
00020     pos = 0;
00021 }
00022
00023 unsigned int RandomAccessExternalEeprom::length() {
00024     return (unsigned int) externalEeprom->getDeviceSize();
00025 }
00026
00027 void RandomAccessExternalEeprom::seek(unsigned int pos) {
00028     this->pos = pos;
00029 }
00030
00031 void RandomAccessExternalEeprom::close() {
00032 }
00033
00034 void RandomAccessExternalEeprom::write(unsigned char* b, int len) {
00035     writeBytes(b, len);
}

void RandomAccessExternalEeprom::write(unsigned char b) {
    writeByte(b);
}

void RandomAccessExternalEeprom::writeByte(unsigned char b) {
    externalEeprom->write(pos++, b);
}

void RandomAccessExternalEeprom::writeBytes(unsigned char* b, int len) {
    for (int i = 0; i < len; i++) {
        externalEeprom->write(pos++, b[i]);
    }
}

void RandomAccessExternalEeprom::writeBoolean(bool v) {
    externalEeprom->write(pos++, (unsigned char) v);
}

void RandomAccessExternalEeprom::writeChar(char c) {
    externalEeprom->write(pos++, (unsigned char) c);
}

void RandomAccessExternalEeprom::writeUnsignedChar(unsigned char c) {
    externalEeprom->write(pos++, (unsigned char) c);
}

void RandomAccessExternalEeprom::writeInt(int v) {
    externalEeprom->write(pos++, (unsigned char) ((v >> 8) & 0xff));
    externalEeprom->write(pos++, (unsigned char) (v & 0xff));
}

void RandomAccessExternalEeprom::writeUnsignedInt(unsigned int v) {
    writeInt((int) v);
}

void RandomAccessExternalEeprom::writeWord(word v) {
    writeInt((int) v);
}

void RandomAccessExternalEeprom::writeLong(long v) {
    externalEeprom->write(pos++, (unsigned char) ((v >> 24) & 0xff));
    externalEeprom->write(pos++, (unsigned char) ((v >> 16) & 0xff));
    externalEeprom->write(pos++, (unsigned char) ((v >> 8) & 0xff));
    externalEeprom->write(pos++, (unsigned char) (v & 0xff));
}

void RandomAccessExternalEeprom::writeUnsignedLong(unsigned long v) {
    writeLong((long) v);
}

void RandomAccessExternalEeprom::writeFloat(float v) {
    writeLong((long) v);
}

void RandomAccessExternalEeprom::writeDouble(double v) {
    writeLong((long) v);
}

unsigned char RandomAccessExternalEeprom::readByte() {
    return (unsigned char) externalEeprom->read(pos++);
}

bool RandomAccessExternalEeprom::readBoolean() {
    return (bool) externalEeprom->read(pos++);
}

char RandomAccessExternalEeprom::readChar() {
    return (char) externalEeprom->read(pos++);
}

unsigned char RandomAccessExternalEeprom::readUnsignedChar() {
    return (unsigned char) externalEeprom->read(pos++);
}

int RandomAccessExternalEeprom::readInt() {
    int v = 0;
    v = externalEeprom->read(pos++);
    v <<= 8;
    v |= (externalEeprom->read(pos++) & 0xff);
    return v;
}
```

```

unsigned int RandomAccessExternalEeprom::readUnsignedInt() {
    return (unsigned int) readInt();
}

word RandomAccessExternalEeprom::readWord() {
    return (word) readInt();
}

long RandomAccessExternalEeprom::readLong() {
    long v = 0;
    v = externalEeprom->read(pos++);
    v <<= 8;
    v |= (externalEeprom->read(pos++) & 0xff);
    v <<= 8;
    v |= (externalEeprom->read(pos++) & 0xff);
    v <<= 8;
    v |= (externalEeprom->read(pos++) & 0xff);
    return v;
}

unsigned long RandomAccessExternalEeprom::readUnsignedLong() {
    return (unsigned long) readLong();
}

float RandomAccessExternalEeprom::readFloat() {
    return (float) readLong();
}

double RandomAccessExternalEeprom::readDouble() {
    return (double) readLong();
}

void RandomAccessExternalEeprom::readFully(unsigned char* b, int len)
{
    for (int i = 0; i < len; i++) {
        b[i] = externalEeprom->read(pos++);
    }
}

unsigned int RandomAccessExternalEeprom::skipBytes(unsigned int n) {
    unsigned int skipped;
    unsigned int newpos;
    newpos = pos + n;
    if (newpos > length()) {
        newpos = length();
    }
    skipped = newpos - pos;
    pos = newpos;
    return skipped;
}

#endif /* __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_CPP__ */
```

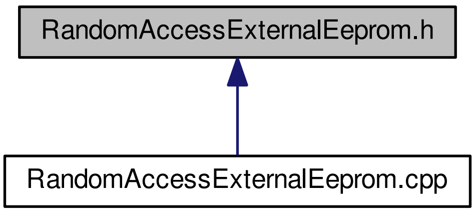
5.95 RandomAccessExternalEeprom.h File Reference

```
#include <Arduino.h>
#include <RandomAccess.h>
#include <Closeable.h>
#include <ExternalEeprom.h>
```

Include dependency graph for RandomAccessExternalEeprom.h:



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



Classes

- class RandomAccessExternalEeprom

5.96 RandomAccessExternalEeprom.h

```
#ifndef __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_H__
#define __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_H__ 1

#include <Arduino.h>
#include <RandomAccess.h>
#include <Closeable.h>
#include <ExternalEeprom.h>

class RandomAccessExternalEeprom : public RandomAccess, public
      virtual Closeable {

    ExternalEeprom* externalEeprom;

    unsigned int pos;

public:

    RandomAccessExternalEeprom(ExternalEeprom* externalEeprom);

    virtual void seek(unsigned int pos);

    unsigned int length();

    virtual void close();

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char b);

    virtual void writeByte(unsigned char b);

    virtual void writeBytes(unsigned char* b, int len);

    virtual void writeBoolean(bool v);

    virtual void writeChar(char c);

    virtual void writeUnsignedChar(unsigned char c);

    virtual void writeInt(int v);

    virtual void writeUnsignedInt(unsigned int v);

    virtual void writeWord(word v);

    virtual void writeLong(long v);

    virtual void writeUnsignedLong(unsigned long v);

    virtual void writeFloat(float v);

    virtual void writeDouble(double v);

    virtual unsigned char readByte();

    virtual bool readBoolean();

    virtual char readChar();

    virtual unsigned char readUnsignedChar();

    virtual int readInt();

    virtual unsigned int readUnsignedInt();

    virtual word readWord();

    virtual long readLong();

    virtual unsigned long readUnsignedLong();

    virtual float readFloat();

    virtual double readDouble();

    virtual void readFully(unsigned char* b, int len);

    virtual unsigned int skipBytes(unsigned int n);
};
#endif /* __ARDUINO_IO_RANDOM_ACCESS_EXTERNAL_EEPROM_H__ */
```

5.97 RandomAccessResource.cpp File Reference

Macros

- #define __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_CPP__ 1

5.97.1 Macro Definition Documentation

5.97.1.1 #define __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_CPP__ 1

Araduino IO.

RandomAccessResource

Instances of this class support both reading and writing to a random access resource. A random access resource behaves like a large array of bytes stored in the resource system.

Definition at line 12 of file RandomAccessResource.cpp.

5.98 RandomAccessResource.cpp

```
00001
00011 #ifndef __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_CPP__
00012 #define __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_CPP__ 1
00013
00014 #if USING_RESOURCE_LIBRARIES
00015
00016 #include "RandomAccessResource.h"
00017
00018 RandomAccessResource::RandomAccessResource(Resource* resource) : resource(resource) {
00019 }
00020
00021 unsigned int RandomAccessResource::length() {
00022     return (unsigned int) resource->size();
00023 }
00024
00025 void RandomAccessResource::seek(unsigned int pos) {
00026     resource->seek(0, pos);
00027 }
00028
00029 void RandomAccessResource::close() {
00030     resource->close();
00031 }
00032
00033 void RandomAccessResource::write(unsigned char* b, int len) {
00034     writeBytes(b, len);
00035 }
00036
00037 void RandomAccessResource::write(unsigned char b) {
00038     writeByte(b);
00039 }
00040
00041 void RandomAccessResource::writeByte(unsigned char b) {
00042     resource->write(b);
00043 }
00044
00045 void RandomAccessResource::writeBytes(unsigned char* b, int len) {
00046     for (int i = 0; i < len; i++) {
00047         resource->write(b[i]);
00048     }
00049 }
00050
00051 void RandomAccessResource::writeBoolean(bool v) {
00052     resource->write((unsigned char) v);
00053 }
00054
00055 void RandomAccessResource::writeChar(char c) {
00056     resource->write((unsigned char) c);
00057 }
00058
00059 void RandomAccessResource::writeUnsignedChar(unsigned char c) {
00060     resource->write((unsigned char) c);
00061 }
00062
00063 void RandomAccessResource::writeInt(int v) {
00064     resource->write((unsigned char) ((v >> 8) & 0xff));
00065     resource->write((unsigned char) (v & 0xff));
00066 }
00067
00068 void RandomAccessResource::writeUnsignedInt(unsigned int v) {
00069     writeInt((int) v);
00070 }
00071
00072 void RandomAccessResource::writeWord(word v) {
00073     writeInt((int) v);
00074 }
00075
00076 void RandomAccessResource::writeLong(long v) {
00077     resource->write((unsigned char) ((v >> 24) & 0xff));
00078     resource->write((unsigned char) ((v >> 16) & 0xff));
00079     resource->write((unsigned char) ((v >> 8) & 0xff));
00080     resource->write((unsigned char) (v & 0xff));
00081 }
00082
00083 void RandomAccessResource::writeUnsignedLong(unsigned long v) {
00084     writeLong((long) v);
00085 }
00086
00087 void RandomAccessResource::writeFloat(float v) {
    writeLong((long) v);
}

void RandomAccessResource::writeDouble(double v) {
    writeLong((long) v);
}

unsigned char RandomAccessResource::readByte() {
    return (unsigned char) resource->read();
}

bool RandomAccessResource::readBoolean() {
    return (bool) resource->read();
}

char RandomAccessResource::readChar() {
    return (char) resource->read();
}

unsigned char RandomAccessResource::readUnsignedChar() {
    return (unsigned char) resource->read();
}

int RandomAccessResource::readInt() {
    int v = 0;
    v = resource->read();
    v <<= 8;
    v |= (resource->read() & 0xff);
    return v;
}

unsigned int RandomAccessResource::readUnsignedInt() {
    return (unsigned int) readInt();
}

word RandomAccessResource::readWord() {
    return (word) readInt();
}

long RandomAccessResource::readLong() {
    long v = 0;
    v = resource->read();
    v <<= 8;
    v |= (resource->read() & 0xff);
    v <<= 8;
    v |= (resource->read() & 0xff);
    v <<= 8;
    v |= (resource->read() & 0xff);
    return v;
}

unsigned long RandomAccessResource::readUnsignedLong() {
    return (unsigned long) readLong();
}

float RandomAccessResource::readFloat() {
    return (float) readLong();
}

double RandomAccessResource::readDouble() {
    return (double) readLong();
}

void RandomAccessResource::readFully(unsigned char* b, int len) {
    for (int i = 0; i < len; i++) {
        b[i] = resource->read();
    }
}

unsigned int RandomAccessResource::skipBytes(unsigned int n) {
    unsigned int pos;
    unsigned int len;
    unsigned int newpos;
    pos = (unsigned int) resource->tell();
    len = resource->size();
    newpos = pos + n;
    if (newpos > len) {
        newpos = len;
    }
    seek(newpos);
    return (unsigned int) (newpos - pos);
}

#endif /* USING_RESOURCE_LIBRARIES */

#endif /* __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_CPP__ */
```

5.99 RandomAccessResource.h File Reference

5.100 RandomAccessResource.h

```

#ifndef __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_H__
#define __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_H__ 1

#if USING_RESOURCE_LIBRARIES

#include <RandomAccess.h>
#include <Closeable.h>
#include <Resource.h>

class RandomAccessResource : public RandomAccess {

    Resource* resource;

public:

    RandomAccessResource(Resource* resource);

    virtual void seek(unsigned int pos);

    unsigned int length();

    virtual void close();

    virtual void write(unsigned char* b, int len);

    virtual void write(unsigned char b);

    virtual void writeByte(unsigned char b);

    virtual void writeBytes(unsigned char* b, int len);

    virtual void writeBoolean(bool v);

    virtual void writeChar(char c);

    virtual void writeUnsignedChar(unsigned char c);

    virtual void writeInt(int v);

    virtual void writeUnsignedInt(unsigned int v);

    virtual void writeWord(word v);

    virtual void writeLong(long v);

    virtual void writeUnsignedLong(unsigned long v);

    virtual void writeFloat(float v);

    virtual void writeDouble(double v);

    virtual unsigned char readByte();

    virtual bool readBoolean();

    virtual char readChar();

    virtual unsigned char readUnsignedChar();

    virtual int readInt();

    virtual unsigned int readUnsignedInt();

    virtual word readWord();

    virtual long readLong();

    virtual unsigned long readUnsignedLong();

    virtual float readFloat();

    virtual double readDouble();

    virtual void readFully(unsigned char* b, int len);

    virtual unsigned int skipBytes(unsigned int n);
};

#endif /* USING_RESOURCE_LIBRARIES */

```

```
00251 #endif /* __ARDUINO_IO_RANDOM_ACCESS_RESOURCE_H__ */
```

5.101 ResourceInputStream.cpp File Reference

Macros

- #define __ARDUINO_IO_RESOURCE_INPUT_STREAM_CPP__ 1

5.101.1 Macro Definition Documentation

5.101.1.1 #define __ARDUINO_IO_RESOURCE_INPUT_STREAM_CPP__ 1

Arduino IO.

ResourceInputStream

A ResourceInputStream obtains input bytes from a resource in a file system.

Definition at line 10 of file ResourceInputStream.cpp.

5.102 ResourceInputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_RESOURCE_INPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_RESOURCE_INPUT_STREAM_CPP__ 1
00011
00012 #if USING_RESOURCE_LIBRARIES
00013
00014 #include "ResourceInputStream.h"
00015
00016 ResourceInputStream::ResourceInputStream(Resource* resource) : resource(resource) {
00017     markpos = 0;
00018     pos = 0;
00019     resourceSize = resource->size();
00020     resource->rewind();
00021 }
00022
00023 int ResourceInputStream::available() {
00024     if ((resourceSize - pos) > 0) {
00025         return 1;
00026     }
00027     return 0;
00028 }
00029
00030 void ResourceInputStream::close() {
00031     resource->close();
00032 }
00033
00034 void ResourceInputStream::mark() {
00035     markpos = pos;
00036 }
00037
00038 bool ResourceInputStream::markSupported() {
00039     return true;
00040 }
00041
00042 int ResourceInputStream::read() {
00043     if (resource->eor()) {
00044         pos = resourceSize;
00045         return -1;
00046     }
00047     pos++;
00048     return (int) resource->read();
00049 }
00050
00051 void ResourceInputStream::reset() {
00052     resource->seek((Resource::ResourceSeekOrigin)0, markpos);
00053 }
00054
00055 #endif /* USING_RESOURCE_LIBRARIES */
00056
00057 #endif /* __ARDUINO_IO_RESOURCE_INPUT_STREAM_CPP__ */
```

5.103 ResourceInputStream.h File Reference

5.104 ResourceInputStream.h

```
#ifndef __ARDUINO_IO_RESOURCE_INPUT_STREAM_H__
#define __ARDUINO_IO_RESOURCE_INPUT_STREAM_H__ 1

#if USING_RESOURCE_LIBRARIES

#include <InputStream.h>
#include <Resource.h>

class ResourceInputStream : public virtual InputStream {
protected:

    /*
     * The resource where data is stored.
     */
    Resource* resource;

    /*
     * Current position
     */
    unsigned int pos;

    /*
     * The currently marked position in the stream.
     */
    unsigned int markpos;

    /*
     * The size of the resource.
     */
    unsigned int resourceSize;

public:

    ResourceInputStream(Resource* resource);

    virtual int available();

    virtual void close();

    virtual void mark();

    virtual bool markSupported();

    using InputStream::read;

    virtual int read();

    virtual void reset();
};

#endif /* USING_RESOURCE_LIBRARIES */

#endif /* __ARDUINO_IO_RESOURCE_INPUT_STREAM_H__ */
```

5.105 ResourceOutputStream.cpp File Reference

Macros

- #define __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_CPP__ 1

5.105.1 Macro Definition Documentation

5.105.1.1 #define __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_CPP__ 1

Arduino IO.

ResourceOutputStream

A resource output stream is an output stream for writing data to a Resource.

Definition at line 10 of file ResourceOutputStream.cpp.

5.106 ResourceOutputStream.cpp

```

#ifndef __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_CPP__
#define __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_CPP__ 1

#if USING_RESOURCE_LIBRARIES

#include "ResourceOutputStream.h"

ResourceOutputStream::ResourceOutputStream(Resource* resource) : resource(resource) {
}

void ResourceOutputStream::close() {
    resource->close();
}

void ResourceOutputStream::write(unsigned char b) {
    resource->write(b);
}

#endif /* USING_RESOURCE_LIBRARIES */

#endif /* __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_CPP__ */
```

5.107 ResourceOutputStream.h File Reference

5.108 ResourceOutputStream.h

```

#ifndef __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_H__
#define __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_H__ 1

#if USING_RESOURCE_LIBRARIES

#include <OutputStream.h>
#include <Resource.h>

class ResourceOutputStream : public OutputStream {
protected:

    /*
     * The resource where data is stored.
     */
    Resource* resource;

public:

    ResourceOutputStream(Resource* resource);

    virtual void close();

    using OutputStream::write;

    virtual void write(unsigned char b);
};

#endif /* USING_RESOURCE_LIBRARIES */

#endif /* __ARDUINO_IO_RESOURCE_OUTPUT_STREAM_H__ */
```

5.109 ResourceSeekableInputStream.cpp File Reference

Macros

- #define __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_CPP__ 1

5.109.1 Macro Definition Documentation

5.109.1.1 #define __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_CPP__ 1

Arduino IO.

ResourceSeekableInputStream

A ResourceSeekableInputStream obtains input bytes from a resource in a file system that implements Seekable↩ InputStream interface.

Definition at line 11 of file ResourceSeekableInputStream.cpp.

5.110 ResourceSeekableInputStream.cpp

```
00001
00010 #ifndef __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_CPP__
00011 #define __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_CPP__ 1
00012
00013 #if USING_RESOURCE_LIBRARIES
00014
00015 #include "ResourceSeekableInputStream.h"
00016
00017 ResourceSeekableInputStream::ResourceSeekableInputStream(Resource* resource) : ResourceInputStream(resource
      ) {
00018 }
00019
00020 void ResourceSeekableInputStream::seek(unsigned int pos) {
00021     resource->seek((Resource::ResourceSeekOrigin)0, pos);
00022 }
00023
00024 #endif /* USING_RESOURCE_LIBRARIES */
00025
00026 #endif /* __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_CPP__ */
```

5.111 ResourceSeekableInputStream.h File Reference

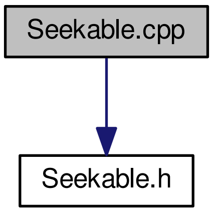
5.112 ResourceSeekableInputStream.h

```

#ifndef __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_H__
#define __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_H__ 1

#if USING_RESOURCE_LIBRARIES

#include <SeekableInputStream.h>
#include <ResourceInputStream.h>
#include <Resource.h>

class ResourceSeekableInputStream : public ResourceInputStream, public
      SeekableInputStream {
public:

    ResourceSeekableInputStream(Resource* resource);

    virtual void seek(unsigned int pos);
};

#endif /* USING_RESOURCE_LIBRARIES */

#endif /* __ARDUINO_IO_RESOURCE_SEEKABLE_INPUT_STREAM_H__ */
```

5.113 Seekable.cpp File Reference

```
#include "Seekable.h"
```

Include dependency graph for Seekable.cpp:



Macros

- #define __ARDUINO_IO_SEEKABLE_CPP__ 1

5.113.1 Macro Definition Documentation

5.113.1.1 #define __ARDUINO_IO_SEEKABLE_CPP__ 1

Arduino IO.

Seekable

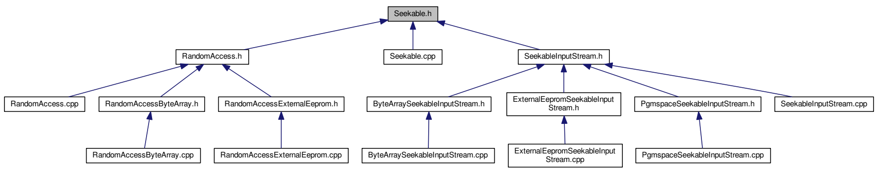
Definition at line 8 of file Seekable.cpp.

5.114 Seekable.cpp

```
00001
00007 #ifndef __ARDUINO_IO_SEEKABLE_CPP__
00008 #define __ARDUINO_IO_SEEKABLE_CPP__ 1
00009
00010 #include "Seekable.h"
00011
00012 #endif /* __ARDUINO_IO_SEEKABLE_CPP__ */
```

5.115 Seekable.h File Reference

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


Seekable.h
RandomAccess.h
Seekable.cpp
SeekableInputStream.h
RandomAccess.cpp
RandomAccessByteArray.h
RandomAccessExternalEeprom.h
ByteArraySeekableInputStream.h
ExternalEepromSeekableInput Stream.h
PgmspaceSeekableInputStream.h
SeekableInputStream.cpp
RandomAccessByteArray.cpp
RandomAccessExternalEeprom.cpp
ByteArraySeekableInputStream.cpp
ExternalEepromSeekableInput Stream.cpp
PgmspaceSeekableInputStream.cpp

Classes

- class Seekable

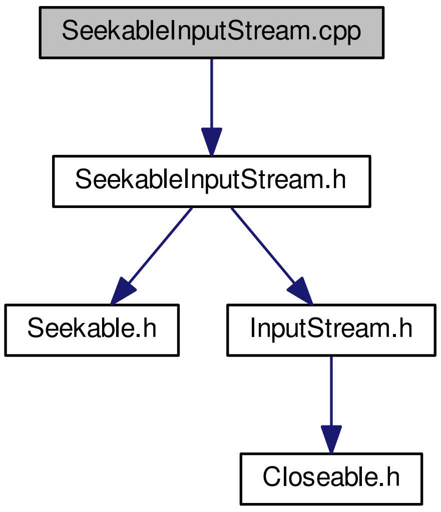
5.116 Seekable.h

```

#ifndef __ARDUINO_IO_SEEKABLE_H__
#define __ARDUINO_IO_SEEKABLE_H__ 1

class Seekable {
public:

    virtual void seek(unsigned int pos) = 0;
};

#endif /* __ARDUINO_IO_SEEKABLE_H__ */
```

5.117 SeekableInputStream.cpp File Reference

`#include "SeekableInputStream.h"`

Include dependency graph for SeekableInputStream.cpp:



Macros

- #define __ARDUINO_IO_SEEKABLE_INPUT_STREAM_CPP__ 1

5.117.1 Macro Definition Documentation

5.117.1.1 #define __ARDUINO_IO_SEEKABLE_INPUT_STREAM_CPP__ 1

Arduino IO.

SeekableInputStream

Definition at line 8 of file SeekableInputStream.cpp.

5.118 SeekableInputStream.cpp

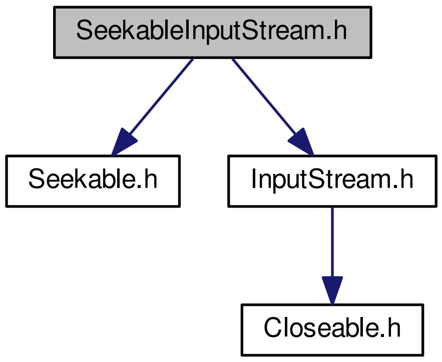
```

```

```
#ifndef __ARDUINO_IO_SEEKABLE_INPUT_STREAM_CPP__
#define __ARDUINO_IO_SEEKABLE_INPUT_STREAM_CPP__ 1

#include "SeekableInputStream.h"

#endif /* __ARDUINO_IO_SEEKABLE_INPUT_STREAM_CPP__ */
```

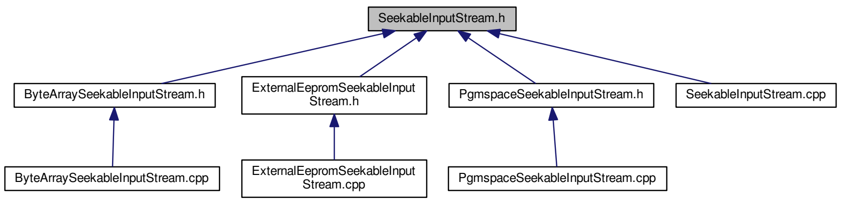
5.119 SeekableInputStream.h File Reference

```
#include <Seekable.h>
#include <InputStream.h>
```

Include dependency graph for SeekableInputStream.h:



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


SeekableInputStream.h
ByteArraySeekableInputStream.h
ExternalEepromSeekableInputStream.h
PgmspaceSeekableInputStream.h
SeekableInputStream.cpp
ByteArraySeekableInputStream.cpp
ExternalEepromSeekableInputStream.cpp
PgmspaceSeekableInputStream.cpp

Classes

- class SeekableInputStream

5.120 SeekableInputStream.h

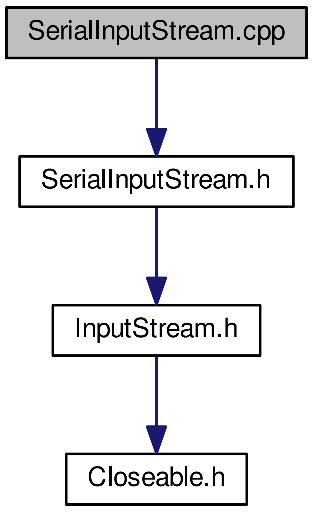
```

#ifndef __ARDUINO_IO_SEEKABLE_INPUT_STREAM_H__
#define __ARDUINO_IO_SEEKABLE_INPUT_STREAM_H__ 1

#include <Seekable.h>
#include <InputStream.h>

class SeekableInputStream : public virtual Seekable, public virtual
```

```
InputStream {
public:

};

#endif /* __ARDUINO_IO_SEEKABLE_INPUT_STREAM_H__ */
```

5.121 SerialInputStream.cpp File Reference

`#include "SerialInputStream.h"`

Include dependency graph for SerialInputStream.cpp:



Macros

- #define __ARDUINO_IO_SERIAL_INPUT_STREAM_CPP__ 1

5.121.1 Macro Definition Documentation

5.121.1.1 #define __ARDUINO_IO_SERIAL_INPUT_STREAM_CPP__ 1

Arduino IO.

SerialInputStream

A SerialInputStream obtains input bytes from a serial port.

Definition at line 10 of file SerialInputStream.cpp.

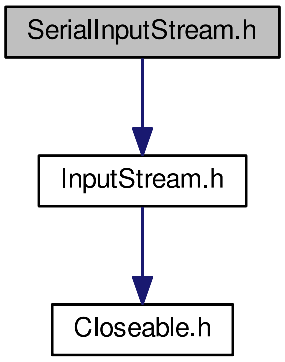
5.122 SerialInputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_SERIAL_INPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_SERIAL_INPUT_STREAM_CPP__ 1
00011
00012 #include "SerialInputStream.h"
00013
00014 #endif /* __ARDUINO_IO_SERIAL_INPUT_STREAM_CPP__ */
```

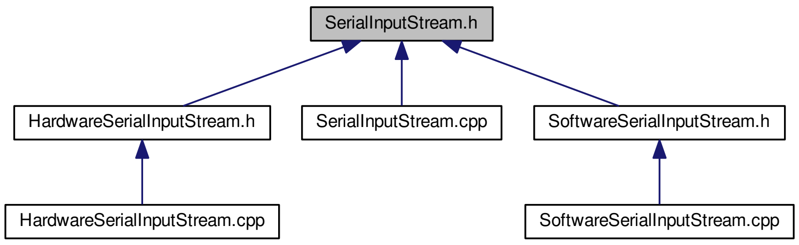
5.123 SerialInputStream.h File Reference

`#include <InputStream.h>`

Include dependency graph for SerialInputStream.h:



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


SerialInputStream.h
HardwareSerialInputStream.h
SerialInputStream.cpp
SoftwareSerialInputStream.h
HardwareSerialInputStream.cpp
SoftwareSerialInputStream.cpp

Classes

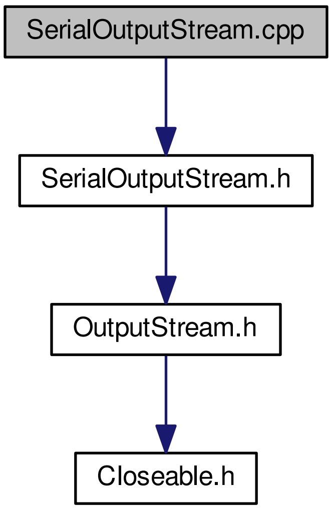
- class SerialInputStream

5.124 SerialInputStream.h

```

#ifndef __ARDUINO_IO_SERIAL_INPUT_STREAM_H__
#define __ARDUINO_IO_SERIAL_INPUT_STREAM_H__ 1

#include <InputStream.h>

class SerialInputStream : public InputStream {
};

#endif /* __ARDUINO_IO_SERIAL_INPUT_STREAM_H__ */
```

5.125 SerialOutputStream.cpp File Reference

`#include <SerialOutputStream.h>`
Include dependency graph for SerialOutputStream.cpp:



Macros

- #define __ARDUINO_IO_SERIAL_OUTPUT_STREAM_CPP__ 1

5.125.1 Macro Definition Documentation

5.125.1.1 #define __ARDUINO_IO_SERIAL_OUTPUT_STREAM_CPP__ 1

Arduino IO.

SerialOutputStream

A serial output stream is a output stream to write in a serial port.

Definition at line 10 of file SerialOutputStream.cpp.

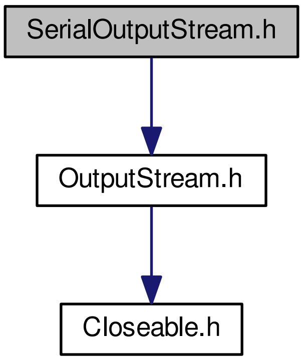
5.126 SerialOutputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_SERIAL_OUTPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_SERIAL_OUTPUT_STREAM_CPP__ 1
00011
00012 #include <SerialOutputStream.h>
00013
00014 #endif /* __ARDUINO_IO_SERIAL_OUTPUT_STREAM_CPP__ */
```

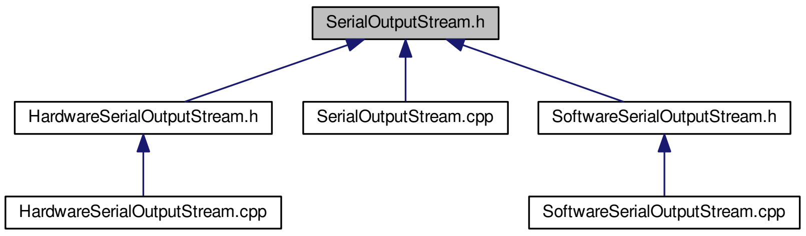
5.127 SerialOutputStream.h File Reference

`#include <OutputStream.h>`

Include dependency graph for SerialOutputStream.h:



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


SerialOutputStream.h
HardwareSerialOutputStream.h
SerialOutputStream.cpp
SoftwareSerialOutputStream.h
HardwareSerialOutputStream.cpp
SoftwareSerialOutputStream.cpp

Classes

- class SerialOutputStream

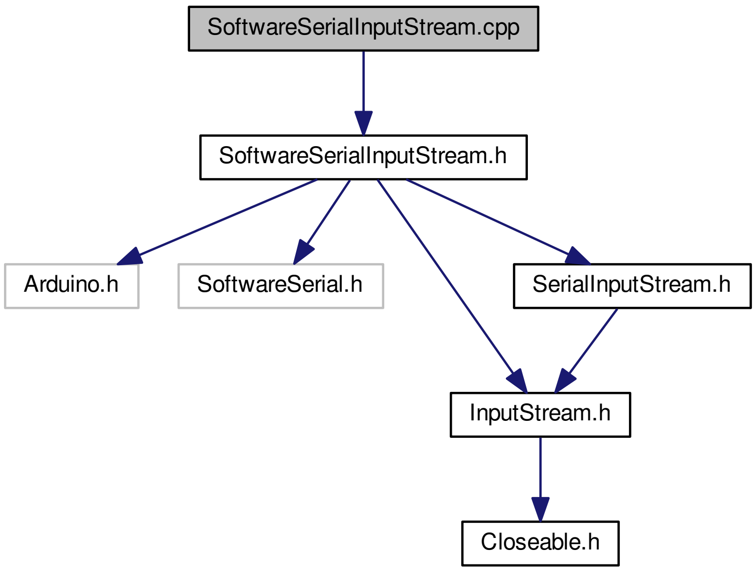
5.128 SerialOutputStream.h

```
#ifndef __ARDUINO_IO_SERIAL_OUTPUT_STREAM_H__
#define __ARDUINO_IO_SERIAL_OUTPUT_STREAM_H__ 1

#include <OutputStream.h>

class SerialOutputStream : public OutputStream {
};

#endif /* __ARDUINO_IO_SERIAL_OUTPUT_STREAM_H__ */
```

5.129 SoftwareSerialInputStream.cpp File Reference

`#include "SoftwareSerialInputStream.h"`

Include dependency graph for SoftwareSerialInputStream.cpp:



Macros

- #define __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_CPP__ 1

5.129.1 Macro Definition Documentation

5.129.1.1 #define __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_CPP__ 1

Arduino IO.

SoftwareSerialInputStream

A SoftwareSerialInputStream obtains input bytes from a serial port.

Definition at line 10 of file SoftwareSerialInputStream.cpp.

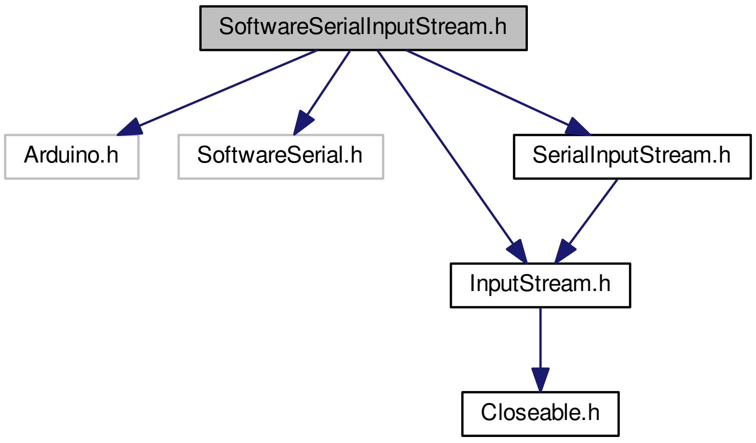
5.130 SoftwareSerialInputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_CPP__ 1
00011
00012 #include "SoftwareSerialInputStream.h"
00013
00014 SoftwareSerialInputStream::SoftwareSerialInputStream(
      SoftwareSerial *softwareSerial,
00015         unsigned int boudRate) :
00016         softwareSerial(softwareSerial) {
00017     softwareSerial->begin(boudRate);
00018 }
00019
00020 int SoftwareSerialInputStream::available() {
00021     softwareSerial->available();
00022 }
00023
00024 int SoftwareSerialInputStream::read() {
00025     return softwareSerial->read();
00026 }
00027
00028 #endif /* __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_CPP__ */
```

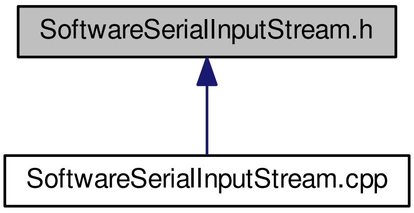
5.131 SoftwareSerialInputStream.h File Reference

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <InputStream.h>
#include <SerialInputStream.h>
```

Include dependency graph for SoftwareSerialInputStream.h:



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



Classes

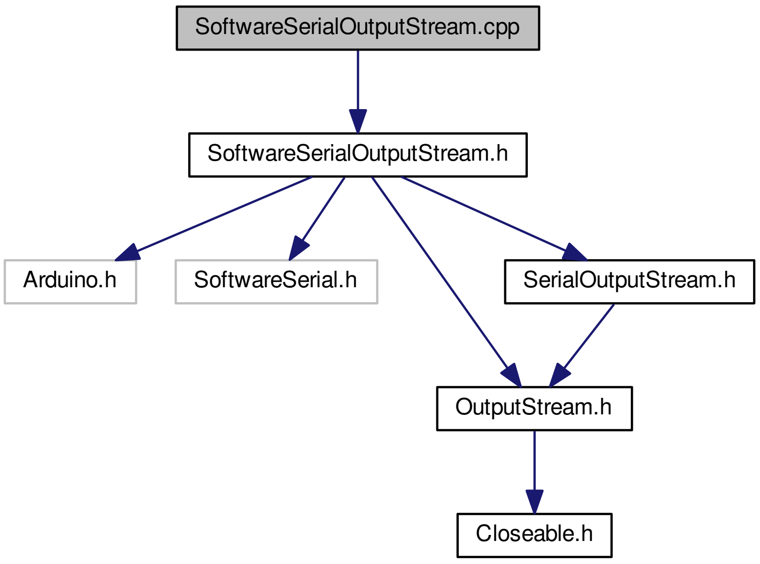
- class SoftwareSerialInputStream

5.132 SoftwareSerialInputStream.h

```

#ifndef __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_H__
#define __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_H__ 1

#include <Arduino.h>
#include <SoftwareSerial.h>
#include <InputStream.h>
#include <SerialInputStream.h>

class SoftwareSerialInputStream : public
SerialInputStream {
protected:

    SoftwareSerial *softwareSerial;

public:

    SoftwareSerialInputStream(SoftwareSerial *softwareSerial, unsigned int
boudRate);

    virtual int available();

    virtual int read();
};

#endif /* __ARDUINO_IO_SOFTWARE_SERIAL_INPUT_STREAM_H__ */
```

5.133 SoftwareSerialOutputStream.cpp File Reference

`#include "SoftwareSerialOutputStream.h"`
Include dependency graph for SoftwareSerialOutputStream.cpp:



Macros

- #define __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_CPP__ 1

5.133.1 Macro Definition Documentation

5.133.1.1 #define __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_CPP__ 1

Arduino IO.

SoftwareSerialOutputStream

A software serial output stream is a output stream to write in a serial port.

Definition at line 10 of file SoftwareSerialOutputStream.cpp.

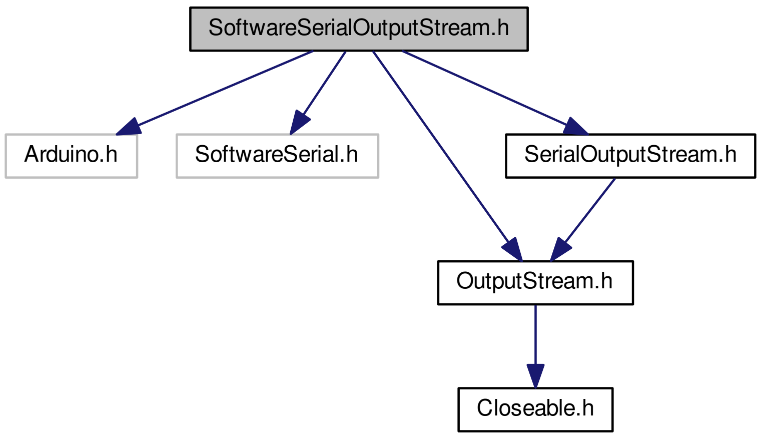
5.134 SoftwareSerialOutputStream.cpp

```
00001
00009 #ifndef __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_CPP__ 1
00011
00012 #include "SoftwareSerialOutputStream.h"
00013
00014 SoftwareSerialOutputStream::SoftwareSerialOutputStream
      (SoftwareSerial *serial,
00015         unsigned int boudRate) :
00016         softwareSerial(serial) {
00017     serial->begin(boudRate);
00018 }
00019
00020 void SoftwareSerialOutputStream::write(unsigned char b) {
00021     softwareSerial->write(b);
00022 }
00023
00024 #endif /* __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_CPP__ */
```

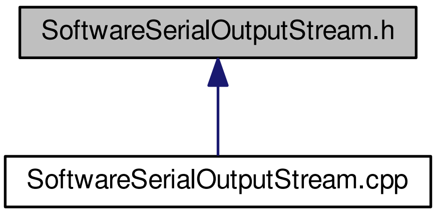
5.135 SoftwareSerialOutputStream.h File Reference

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <OutputStream.h>
#include <SerialOutputStream.h>
```

Include dependency graph for SoftwareSerialOutputStream.h:



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



Classes

- class SoftwareSerialOutputStream

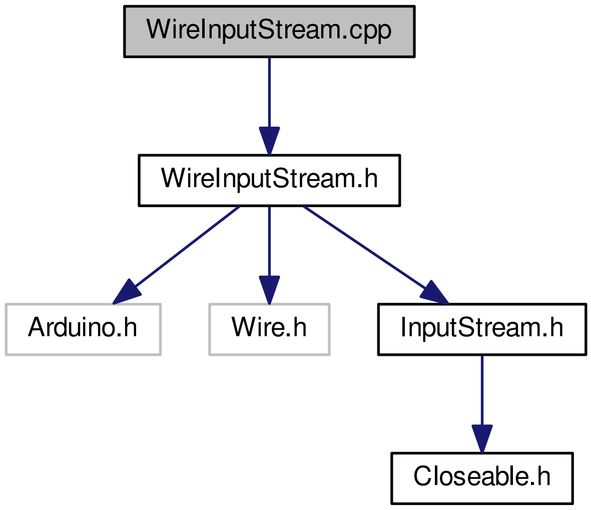
5.136 SoftwareSerialOutputStream.h

```
#ifndef __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_H__
#define __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_H__ 1

#include <Arduino.h>
#include <SoftwareSerial.h>
#include <OutputStream.h>
#include <SerialOutputStream.h>

class SoftwareSerialOutputStream : public
SerialOutputStream {
protected:

    /*
     * The software serial where data is written.
     */
    SoftwareSerial *softwareSerial;

public:

    SoftwareSerialOutputStream(SoftwareSerial *serial, unsigned int boudRate);

    virtual void write(unsigned char b);
};

#endif /* __ARDUINO_IO_SOFTWARE_SERIAL_OUTPUT_STREAM_H__ */
```

5.137 WireInputStream.cpp File Reference

```
#include "WireInputStream.h"
```

Include dependency graph for WireInputStream.cpp:



Macros

- #define __ARDUINO_IO_WIRE_INPUT_STREAM_CPP__ 1

5.137.1 Macro Definition Documentation

5.137.1.1 #define __ARDUINO_IO_WIRE_INPUT_STREAM_CPP__ 1

Arduino IO.

WireInputStream

A WireInputStream obtains input bytes from the wire bus.

Definition at line 10 of file WireInputStream.cpp.

5.138 WireInputStream.cpp

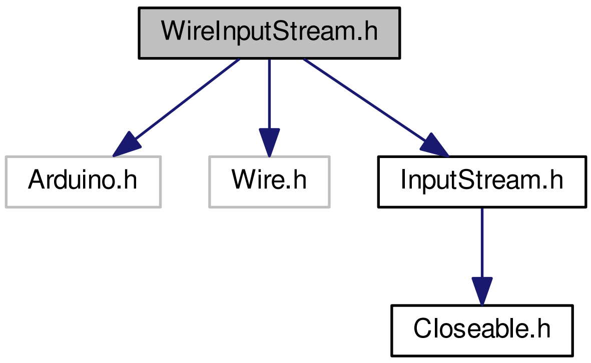
```
00001
00009 #ifndef __ARDUINO_IO_WIRE_INPUT_STREAM_CPP__
00010 #define __ARDUINO_IO_WIRE_INPUT_STREAM_CPP__ 1
00011
00012 #include "WireInputStream.h"
00013
00014 WireInputStream::WireInputStream(unsigned char address) {
00015     this->address = address;
00016     Wire.begin();
00017 }
00018
00019 int WireInputStream::available() {
00020     return Wire.available();
00021 }
00022
00023 int WireInputStream::read() {
00024     Wire.beginTransmission(address);
00025     Wire.write((unsigned char) (address & 0xff));
00026     Wire.endTransmission();
00027     Wire.requestFrom(address, (unsigned char) 1);
```

```
    while (!Wire.available())
        ;
    return Wire.read();
}

int WireInputStream::read(unsigned char* b, int off, int len) {
    int i;
    Wire.beginTransmission(address);
    Wire.write((unsigned char) (address & 0xff));
    Wire.endTransmission();
    Wire.requestFrom(address, (int) len);
    for (i = 0; i < len; i++) {
        while (!Wire.available())
            ;
        b[off + i] = (unsigned char) Wire.read();
    }
    return i;
}

#endif /* __ARDUINO_IO_WIRE_INPUT_STREAM_CPP__ */
```

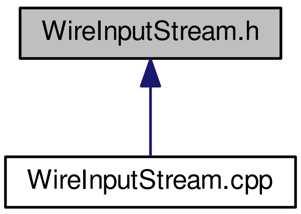
5.139 WireInputStream.h File Reference

```
#include <Arduino.h>
#include <Wire.h>
#include <InputStream.h>
```

Include dependency graph for WireInputStream.h:



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



Classes

- class WireInputStream

5.140 WireInputStream.h

```
#ifndef __ARDUINO_IO_WIRE_INPUT_STREAM_H__
#define __ARDUINO_IO_WIRE_INPUT_STREAM_H__ 1

#include <Arduino.h>
#include <Wire.h>
#include <InputStream.h>

class WireInputStream : public InputStream {
protected:

    unsigned char address;

public:

    WireInputStream(unsigned char addredd);

    virtual int available();

    virtual int read();

    virtual int read(unsigned char* b, int off, int len);
};

#endif /* __ARDUINO_IO_WIRE_INPUT_STREAM_H__ */
```

5.141 WireOutputStream.cpp File Reference

5.142 WireOutputStream.cpp

5.143 WireOutputStream.h File Reference

5.144 WireOutputStream.h

Index