bitstream

Paul Long

Table of Contents

bitstream

The Boost.Bitstream library provides a succinct means to read and write binary data using the extraction and insertion operators. Its semantics mirror `std::stringstream`, it supports big- and little-endian integrals, can be extended to support user-defined types, and its performance is comparable to hand-coded codecs.

Tutorial

This tutorial introduces `bitstream` by showing how to write a `bool` followed by an `int` to a buffer and then reading it back.

All of the bitstream classes can be used by including the `bstream.hpp` header file.

```
#include <boost/bitstream/bstream.hpp>
```

There is also an example of writing a codec at

bitstream_codec.cpp.

Testing

Tests are provided at

test_basic.cpp

Acknowledgements

The author thanks

- Paul A. Bristow who produced a draft of documentation and other files.

References

Rationale

This section records the rationale and compromises for some design decisions.

The proposal was iniated by these Boost list discussions.

History

1. Project started by Paul Long June 2013. The proposal was iniated by these Boost list discussions.

2. First Boost Sandbox release for public comment July 2013.

Version Info

Last edit to Quickbook file bitstream.qbk was at 04:33:29 PM on 2013-Jul-22.

Warning

Home page "Last revised" is GMT, not local time. Last edit date is local time.

Boost.Bitstream C++ Reference

Header <boost/bitstream/bstream.hpp>

Bit-stream classes.

This header file contains the bit-stream classes.

Note

See Also:

http://www.boost.org/ for latest version.

http://www.boost.org/libs/bitstream for documentation.

```
namespace boost {
  namespace bitstream {
    class ibitstream;
    class obitstream;
  }
}
```

Class ibitstream

boost::bitstream::ibitstream

Synopsis

```
// In header: <boost/bitstream/bstream.hpp>


class ibitstream : public istream {
public:
  // construct/copy/destruct
  explicit ibitstream(std::ios_base::openmode = std::ios_base::in);
  explicit ibitstream(const char *, std::streamsize = INT_MAX,
                      std::ios_base::openmode = std::ios_base::in);

  // public member functions
  bitbuf * rdbuf() const;
  const char * data() const;
};
```

Description

This class provides an interface to manipulate bits as an input stream.

Note

This class is based on but does not inherit from the standard class, istringstream. The main difference is that this class provides access to bits whereas istringstream provides access to characters.

`ibitstream` public construct/copy/destruct

1.
```
explicit ibitstream(std::ios_base::openmode which = std::ios_base::in);
```

Constructor.

Parameters: `which` Open mode.

2.
```
explicit ibitstream(const char * buffer, std::streamsize size = INT_MAX,
                    std::ios_base::openmode which = std::ios_base::in);
```

Constructor.

Parameters:
`buffer`	Pointer to char array to be accessed.
`size`	Number of accessible bits in char array.
`which`	Open mode.

`ibitstream` public member functions

1.
```
bitbuf * rdbuf() const;
```

Get the bitbuf object associated with the stream upon construction.

Returns: A pointer to the bitbuf object associated with the stream.

2.
```
const char * data() const;
```

Get pointer to current contents of the stream.

Note

This is analogous to istringstream::str().

Returns: Pointer to stream buffer.

Class obitstream

boost::bitstream::obitstream

Synopsis

```
// In header: <boost/bitstream/bstream.hpp>


class obitstream : public ostream {
public:
  // construct/copy/destruct
  explicit obitstream(std::ios_base::openmode = std::ios_base::in);
  explicit obitstream(const char *, std::streamsize = INT_MAX,
                      std::ios_base::openmode = std::ios_base::in);

  // public member functions
  bitbuf * rdbuf() const;
  const char * data() const;
};
```

Description

This class provides an interface to manipulate bits as an output stream.

Note

This class is based on but does not inherit from the standard class, ostringstream. The main difference is that this class provides access to bits whereas ostringstream provides access to characters.

`obitstream` public construct/copy/destruct

1. ```
   explicit obitstream(std::ios_base::openmode which = std::ios_base::in);
   ```

   Constructor.

   Parameters: `which` Open mode.

2. ```
   explicit obitstream(const char * buffer, std::streamsize size = INT_MAX,
                       std::ios_base::openmode which = std::ios_base::in);
   ```

   Constructor.

   Parameters:
   - `buffer` Pointer to char array to be accessed.
   - `size` Number of accessible bits in char array.
   - `which` Open mode.

`obitstream` public member functions

1. ```
   bitbuf * rdbuf() const;
   ```

   Get the bitbuf object associated with the stream upon construction.

   Returns: A pointer to the bitbuf object associated with the stream.

2. ```
   const char * data() const;
   ```

   Get pointer to current contents of the stream.

Note

This is analogous to ostringstream::str().

Returns: Pointer to stream buffer.

Header <boost/bitstream/ibstream.hpp>

This file provides the basic stream classes.

This file provides details about the basic stream classes.

```
namespace boost {
  namespace bitstream {
    class ibstream;
    ibstream & operator>>(ibstream &, bool &);
    ibstream & operator>>(ibstream &, const bool &);
    template<typename T> ibstream & operator>>(ibstream &, T &);
    template<typename T> ibstream & operator>>(ibstream &, const T &);
    template<typename T> ibstream & operator>>(ibstream &, std::vector< T > &);
    template<typename T>
      ibstream & operator>>(ibstream &, std::vector< const T > &);
    template<size_t N> ibstream & operator>>(ibstream &, std::bitset< N > &);
    template<size_t N>
      ibstream & operator>>(ibstream &, const std::bitset< N > &);
  }
}
```

Class ibstream

boost::bitstream::ibstream

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


class ibstream : public boost::bitstream::iob {
public:
  // construct/copy/destruct
  explicit ibstream(bitbuf *);
  template<typename T> friend ibstream & operator>>(ibstream &, const T &);
  template<size_t N>
    friend ibstream & operator>>(ibstream &, const std::bitset< N > &);
  friend ibstream & operator>>(ibstream &, const bool &);

  // public member functions
  std::streamsize gcount() const;
  bitfield gvalue() const;
  bitfield get();
  ibstream & ignore(std::streamsize = 1);
  ibstream & aligng(size_t);
  ibstream & repeat(size_t);
  bool alignedg(size_t);
  bitfield peek();
  ibstream & read(bitfield &, std::streamsize);
  ibstream & readsome(bitfield &, std::streamsize);
  ibstream & seekg(std::streamoff, std::ios_base::seek_dir);
  ibstream & seekg(std::streampos);
  int sync();
  std::streampos tellg();
  ibstream & unget();
};
```

Description

`ibstream` public construct/copy/destruct

1. ```
   explicit ibstream(bitbuf * bb);
   ```

   Constructor.

```
template<typename T> friend ibstream & operator>>(ibstream & ibs, const T & b);
```

Friend const functions for access to badbit().

Get bit field from input stream that must be equal to integral value.

Parameters:	`b`	Integral on right-hand side of operator.
	`ibs`	Reference to ibstream on left-hand side of operator.
Returns:	Reference to ibstream parameter.	

```
template<size_t N>
  friend ibstream & operator>>(ibstream & ibs, const std::bitset< N > & bs);
```

Get bits from input stream that must be equal to bitset value.

Parameters:	`bs`	bitset on right-hand side of operator.
	`ibs`	Reference to ibstream on left-hand side of operator.
Returns:	Reference to ibstream parameter.	
```

```
friend ibstream & operator>>(ibstream & ibs, const bool & b);
```

Get single bit from input stream that must be equal to bool.

Parameters: `b` bool on right-hand side of operator.
`ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

`ibstream` public member functions

1. ```
   std::streamsize gcount() const;
   ```

   Get number of bits extracted by last input operation.

2. ```
   bitfield gvalue() const;
   ```

   Get value extracted by last input operation.

   

   Note

   The value returned by this function is only valid if gcount() returns a value greater than zero.

   Returns: Most recent extracted value.

3. ```
   bitfield get();
   ```

   Get one bit from stream.

   Returns: Next bit from stream.

4. ```
   ibstream & ignore(std::streamsize bits = 1);
   ```

   Ignore, or skip over, bits in stream.

   Parameters: `bits` Number of bits to ignore.
   Returns: This bit stream.

5. ```
   ibstream & aligng(size_t bit);
   ```

   Align get pointer to next bit multiple if not already at one.

   Parameters: `bit` Bit multiple, such as 8 for byte alignment.
   Returns: This bit stream.

6. ```
   ibstream & repeat(size_t repeat);
   ```

   Set repeat count for subsequent vector extractions.

   

   Note

   This function does not extract anything from ibstream. It merely saves a value that any subsequent vector extractions use to know how many bit fields to extract into the same number of vector elements.

Parameters: `repeat` Number of bit fields to extract to each subsequent vector.
Returns: This bit stream.

7.
```
bool alignedg(size_t bit);
```

Determine whether get pointer is aligned to bit multiple.

Example:

```
static const bitset<2> version(0x2);
bitset<4> csrcCount;
bool marker;
bitset<7> payloadType;
DWORD timestamp, ssrcIdentifier;
ibitstream bin(rtpHeader);
bin >> version >> aligng(4) >> csrcCount;
assert(bin.alignedg(8));
// (aligng is redundant here because already at byte alignment)
bin >> aligng(8) >> marker >> payloadType;
bin >> timestamp >> ssrcIdentifier;
```

Parameters: `bit` Bit multiple, such as 8 for byte alignment.
Returns: Whether get pointer is bit-multiple aligned.

8.
```
bitfield peek();
```

Get next bit from stream without advancing get pointer.

Returns: Next bit from stream.

9.
```
ibstream & read(bitfield & value, std::streamsize bits);
```

Get bits from stream.

Note

Unlike get(), this function returns the bit value in an integral, not a bitset.

Parameters: `bits` Number of bits to read.
Returns: This bit stream.

10.
```
ibstream & readsome(bitfield & value, std::streamsize bits);
```

Get "some" bits from stream.

Note

This function is provided for completeness. A bit stream residing in memory does not possess the behavior that, unlike istringstream, would make readsome() behave any different from read().

Parameters: `bits` Number of bits to read.
Returns: This bit stream.

11.
```
ibstream & seekg(std::streamoff offset, std::ios_base::seek_dir dir);
```

Set position of get pointer relative to indicated internal pointer.

Parameters: `dir` Bit pointer to which offset is applied.
`offset` Relative offset from indicated pointer.
Returns: This bit stream.

12.
```
ibstream & seekg(std::streampos position);
```

Set position of get pointer.

Parameters: `position` Bit position.
Returns: This bit stream.

13.
```
int sync();
```

Synchronize input buffer with source of bits.

Returns: 0 if buffered stream and successful; -1 otherwise.

14.
```
std::streampos tellg();
```

Get position of get pointer.

Returns: Bit position of next bit to be read.

15.
```
ibstream & unget();
```

Move get pointer backwards and return bit at new position.

Returns: This bit stream.

Function operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


ibstream & operator>>(ibstream & ibs, bool & b);
```

Description

Get single bit from input stream and place in bool.

Parameters: `b` bool on right-hand side of operator.
`ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

Function operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>

ibstream & operator>>(ibstream & ibs, const bool & b);
```

Description

Get single bit from input stream that must be equal to bool.

Parameters: `b` bool on right-hand side of operator.
`ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>

template<typename T> ibstream & operator>>(ibstream & ibs, T & b);
```

Description

Get bit field from input stream and place in integral.

Parameters: `b` Integral on right-hand side of operator.
`ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>

template<typename T> ibstream & operator>>(ibstream & ibs, const T & b);
```

Description

Get bit field from input stream that must be equal to integral value.

Parameters: `b` Integral on right-hand side of operator.

	`ibs`	Reference to ibstream on left-hand side of operator.
Returns:		Reference to ibstream parameter.

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


template<typename T>
  ibstream & operator>>(ibstream & ibs, std::vector< T > & v);
```

Description

Get bit fields from input stream and place in integral vector.

Note

Starting with the first element, this function populates existing elements in the vector with bit fields sequentially extracted from the input stream. It does not increase the size of the vector, e.g., with push_back(), because it relies on the existing size of the vector to know how many bit fields to extract.

Parameters:	`ibs`	Reference to ibstream on left-hand side of operator.
	`v`	Integral vector on right-hand side of operator.
Returns:		Reference to ibstream parameter.

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


template<typename T>
  ibstream & operator>>(ibstream & ibs, std::vector< const T > & v);
```

Description

Get bit fields from input stream that must be equal to elements in integral vector.

See Also:

Size note for non-const version of this function.

Parameters:	`ibs`	Reference to ibstream on left-hand side of operator.
	`v`	Integral vector on right-hand side of operator.
Returns:		Reference to ibstream parameter.

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


template<size_t N>
  ibstream & operator>>(ibstream & ibs, std::bitset< N > & bs);
```

Description

Get bits from input stream and place in bitset.

Parameters:	`bs`	bitset on right-hand side of operator.
	`ibs`	Reference to ibstream on left-hand side of operator.
Returns:	Reference to ibstream parameter.	

Function template operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/ibstream.hpp>


template<size_t N>
  ibstream & operator>>(ibstream & ibs, const std::bitset< N > & bs);
```

Description

Get bits from input stream that must be equal to bitset value.

Parameters:	`bs`	bitset on right-hand side of operator.
	`ibs`	Reference to ibstream on left-hand side of operator.
Returns:	Reference to ibstream parameter.	

Header <boost/bitstream/iob.hpp>

Base classes and types for bit streams.

This header file contains base classes and types for the bit-stream hierarchy of classes.

parametric manipulators for bit streams.

Note

See Also:

http://www.boost.org/ for latest version.

http://www.boost.org/libs/bitstream for documentation.

```
namespace boost {
  namespace bitstream {
    class bitbuf;
    class iob;

    decltype(std::bitset< 0 >().to_ulong()) typedef bitfield;
  }
}
```

Class bitbuf

boost::bitstream::bitbuf

Synopsis

```
// In header: <boost/bitstream/iob.hpp>


class bitbuf {
public:
  // construct/copy/destruct
  explicit bitbuf(std::ios_base::openmode = std::ios_base::in|std::ios_base::out);
  bitbuf(const char *, std::streamsize = INT_MAX,
         std::ios_base::openmode = std::ios_base::in|std::ios_base::out);

  // public member functions
  const char * data() const;
  void data(const char *);
  std::streamsize in_avail() const;
  std::streampos pubseekoff(std::streamoff, std::ios_base::seekdir);
  std::streampos pubseekpos(std::streampos);
  bitbuf * pubsetbuf(unsigned char *);
  bitbuf * pubsetbuf(unsigned char *, std::streamsize, std::streampos = 0);
  int pubsync();
  bool sbumpb(bitfield &);
  bool sgetb(bitfield &);
  std::streamsize sgetn(bitfield &, std::streamsize);
  bool snextb(bitfield &);
  bool sungetb(bitfield &);

  // protected member functions
  std::streampos gptr() const;
  std::streampos egptr() const;
  std::streampos eback() const;
  void setg(unsigned char *, std::streampos, std::streampos, std::streampos);
  void gbump(std::streamoff);
  std::streampos seekpos(std::streampos);
  std::streampos seekoff(std::streamoff, std::ios_base::seekdir);
  std::streampos assure_valid_get_pointer(std::streampos);
  bitbuf * setbuf(unsigned char *, std::streamsize, std::streampos);
  int sync();
  std::streamsize xsgetn(bitfield &, std::streamsize);

  // private member functions
  unsigned char * current_byte() const;

  // public data members
  static const int npos;
  static const bitfield eof;
};
```

Description

This class represents contiguous memory, accessed as a sequence of bit fields.

Note

This class is based on but does not inherit from the standard class, stringbuf. The main difference is that this class provides access to bits whereas stringbuf provides access to characters.

Currently, this class can only be used to read bits from a buffer; however, it could be easily be fleshed out for writing bits, too.

`bitbuf` public construct/copy/destruct

1.
```
explicit bitbuf(std::ios_base::openmode which = std::ios_base::in|std::ios_base::out);
```

Constructor.

Parameters: `which` Open mode.

2.
```
bitbuf(const char * buffer, std::streamsize size = INT_MAX,
       std::ios_base::openmode which = std::ios_base::in|std::ios_base::out);
```

Constructor.

Parameters:
`buffer`	Pointer to char array to be accessed.
`size`	Number of accessible bits in char array.
`which`	Open mode.

`bitbuf` public member functions

1.
```
const char * data() const;
```

Get pointer to char-array stream buffer.

Note

This is analogous to stringbuf::str().

<xrefsect>
<xreftitle>Todo</xreftitle>
<xrefdescription>

Should this return const?
</xrefdescription>
</xrefsect>

Returns: Pointer to stream buffer.

2.
```
void data(const char * buffer);
```

Set pointer to char-array stream buffer.

Note

This is analogous to stringbuf::str(x).

<xrefsect>
<xreftitle>Todo</xreftitle>
<xrefdescription>

Assure various pointers, etc. are reset to reflect a new pointer.
</xrefdescription>
</xrefsect>

3.
```
std::streamsize in_avail() const;
```

Number of bits currently available to read.

Returns: Number of readable bits.

4.
```
std::streampos pubseekoff(std::streamoff offset, std::ios_base::seekdir way);
```

Reposition get-next-bit pointer relative to current position.

Parameters: `offset` Signed offset from current position for new position.
`way` From which pointer offset is applied for new position.
Returns: Position after offset applied.

5.
```
std::streampos pubseekpos(std::streampos position);
```

Reposition get-next-bit pointer.

Parameters: `position` New bit position.
Returns: Position after repositioning.

6.
```
bitbuf * pubsetbuf(unsigned char * buffer);
```

Set buffer to access.

Parameters: `buffer` Pointer to char array to be accessed.
Returns: Pointer to this object; NULL if error.

7.
```
bitbuf * pubsetbuf(unsigned char * buffer, std::streamsize size,
                   std::streampos position = 0);
```

Set buffer to access.

Parameters: `buffer` Pointer to char array to be accessed.
`position` Offset of first accessible bit in char array.
`size` Number of accessible bits in char array.
Returns: Pointer to this object; NULL if error.

8.
```
int pubsync();
```

Synchronize stream buffer with input or output device.

Returns: 0 if buffered stream and successful; -1 otherwise.

9.
```
bool sbumpb(bitfield & value);
```

Get current bit and advance get pointer.

Parameters: `value` Current bit before advancing pointer.
Returns: Whether okay - eof has not been encountered.

10.
```
bool sgetb(bitfield & value);
```

Get current bit at get pointer.

Parameters: `value` Current bit.
Returns: Whether okay - eof has not been encountered.

11.
```
std::streamsize sgetn(bitfield & value, std::streamsize size);
```

Get sequence of bits.

Parameters: `size` Number of bits in sequence of bits.
`value` Value of bit field.
Returns: Number of bits read from buffer or zero if error or eof.

12.
```
bool snextb(bitfield & value);
```

Advance get pointer and return next bit.

Parameters: `value` Next bit.
Returns: Whether okay - eof has not been encountered.

13.
```
bool sungetb(bitfield & value);
```

Move get pointer backwards and return bit at new position.

Parameters: `value` Bit before position prior to call.
Returns: Whether okay - eof has not been encountered.

`bitbuf` protected member functions

1.
```
std::streampos gptr() const;
```

Returns bit position within accessible input sequence of next bit to be read.

Note

The "get pointer."

Returns: Next bit position.

2.
```
std::streampos egptr() const;
```

Returns bit position just past last bit in accessible input sequence.

Returns: Position after last bit.

3.
```
std::streampos eback() const;
```

Returns first bit position in accessible input sequence.

Returns: Position of first bit.

4.
```
void setg(unsigned char * buffer, std::streampos gbeg, std::streampos gnext,
          std::streampos gend);
```

Set pointer and offsets that define boundaries of and position within accessible input sequence.

Parameters:
- `buffer` Pointer to char array to be accessed.
- `gbeg` Position of first accessible bit in char array.
- `gend` Position of bit immediately after last accessible bit in char array.
- `gnext` Position of next bit to read.

5.
```
void gbump(std::streamoff offset);
```

Advances the get pointer by specified number of bit positions.

Parameters:
- `offset` Value by which to increase the get pointer.

6.
```
std::streampos seekpos(std::streampos position);
```

Set get pointer to absolute position.

Parameters:
- `position` New absolute position for get pointer.

Returns: New position after get pointer modified or npos if error.

7.
```
std::streampos seekoff(std::streamoff offset, std::ios_base::seekdir way);
```

Set get pointer relative to current position.

Parameters:
- `offset` Amount by which get pointer is adjusted.
- `way` From which pointer offset is applied for new position.

Returns: New position after get pointer modified.

8.
```
std::streampos assure_valid_get_pointer(std::streampos position);
```

Assure that position is within bounds of accessible input sequence.

If bit position is within bounds, use as internal get pointer and return it; otherwise, return npos.

Parameters:
- `position` Candidate for new current position, or get pointer.

9.
```
bitbuf * setbuf(unsigned char * buffer, std::streamsize size,
                std::streampos position);
```

Set buffer to access.

Parameters:
- `buffer` Pointer to char array containing bits to access.
- `position` Offset of first accessible bit in char array.
- `size` Number of accessible bits in char array.

Returns: Pointer to this object; NULL if error.

10.
```
int sync();
```

Synchronize stream buffer with input or output device.

Note

Since we do not currently have an I/O device with which to synchronize - we are unbuffered - this function always returns in error with -1.

Returns: 0 if buffered stream and successful; -1 otherwise.

11.
```
std::streamsize xsgetn(bitfield & value, std::streamsize size);
```

Get sequence of bits.

Parameters: `size` Number of bits in sequence of bits.
`value` Value of bit field.
Returns: Number of bits read from buffer or zero if error or eof.

`bitbuf` private member functions

1.
```
unsigned char * current_byte() const;
```

Get pointer to current byte.

Returns: Pointer to byte containing current bit position (the next bit to read).

`bitbuf` public public data members

1.
```
static const int npos;
```

Represents both an unbounded number of bits and an attempt to move past bit-stream bounds.

Note

A consumer might encounter this value where member functions of this class return a value of streamsize or streampos type.

2.
```
static const bitfield eof;
```

Special, end-of-file value.

Note

This class cannot extract a bit sequence that is equal to this value. Therefore, when a member function returns this value, the consumer should knowthat the end of file has been reached. Actually, an attempt was made to either advance past the end or beginning of the accessible bits in the buffer.

Class iob

boost::bitstream::iob

Synopsis

```
// In header: <boost/bitstream/iob.hpp>


class iob {
public:
  // construct/copy/destruct
  explicit iob(bitbuf *);
  iob();
  ~iob();

  // public member functions
  operator void *() const;
  bool good() const;
  bool eof() const;
  bool fail() const;
  bool bad() const;
  bool operator!() const;
  operator bool() const;
  std::ios_base::iostate rdstate() const;
  void setstate(std::ios_base::iostate);
  void clear(std::ios_base::iostate = std::ios_base::goodbit);
  bitbuf * rdbuf() const;
  bitbuf * rdbuf(bitbuf *);

  // protected member functions
  void init(bitbuf *);
  void badbit();
  void failbit();
  void eofbit();
};
```

Description

Base class for all bit-stream classes.

`iob` public construct/copy/destruct

1. ```
   explicit iob(bitbuf * bb);
   ```

   Constructor.

   Parameters: `bb` Pointer to a bitbuf object.

2. ```
   iob();
   ```

3. ```
   ~iob();
   ```

   Destructor.

`iob` public member functions

1. ```
   operator void *() const;
   ```

   Evaluate stream object for success.

This function returns null if the internal failbit or badbit are set for this stream; otherwise, it returns a non-zero pointer.

Returns: Whether 0 if failbit or badbit are set; non-zero otherwise.

2.
```
bool good() const;
```

Check if bitstream is good for continued operation.

Returns: Whether any of the error flags are set.

3.
```
bool eof() const;
```

Check if error flag, eofbit, is set.

Note

eofbit is set when an operation attempts to access a bit position outside of the bit stream, e.g., reading past end of bitstream.

Returns: Whether previous input operation set eofbit.

4.
```
bool fail() const;
```

Check if either error flag, failbit or badbit, is set.

Note

failbit is set when there is an error with the internal logic of an operation.

See Also:

Note for bad().
Returns: Whether previous input operation set failbit or badbit.

5.
```
bool bad() const;
```

Check if error flag, badbit, is set.

Note

badbit is set if the integrity of the stream is lost, e.g., encountered unexpected value.

Returns: Whether previous input operation set badbit.

6.
```
bool operator!() const;
```

Evaluate stream object for failure.

This function returns whether the internal failbit or badbit has been set for this ibitstream.

Note

Same as calling fail(). bin.fail() is the same as !bin.

Returns: Whether the failbit or badbit has been set.

7.
```
operator bool() const;
```

Evaluate stream object for success.

This function returns whether the internal failbit and badbit are not set for this ibitstream.

Note

Same as calling !fail(). !bin.fail() is the same as bin.

Returns: Whether the failbit and badbit are not set.

8.
```
std::ios_base::iostate rdstate() const;
```

Get error state flags.

Returns: Error state flags.

9.
```
void setstate(std::ios_base::iostate state);
```

Set error state flags.

Note

This function sets states additively - no state is cleared.

Parameters: `state` Error state flags.

10.
```
void clear(std::ios_base::iostate state = std::ios_base::goodbit);
```

Set error state flags.

Parameters: `state` Error state flags.

11.
```
bitbuf * rdbuf() const;
```

Get the bitbuf object currently associated with the stream.

Returns: A pointer to the bitbuf object associated with the stream.

12.
```
bitbuf * rdbuf(bitbuf * bb);
```

Set the bitbuf object associated with the stream.

Returns: A pointer to the bitbuf object previously associated with the stream.

`iob` protected member functions

1. ```
   void init(bitbuf * bb);
   ```

   Initialize member variables.

   Parameters: `bb` Pointer to a bitbuf object.

2. ```
   void badbit();
   ```

   Set badbit error flag.

3. ```
   void failbit();
   ```

   Set failbit error flag.

4. ```
   void eofbit();
   ```

   Set eofbit error flag.

Global bitfield

boost::bitstream::bitfield

Synopsis

```
// In header: <boost/bitstream/iob.hpp>

decltype(std::bitset< 0 >().to_ulong()) typedef bitfield;
```

Description

Integral type for bit-field values.

Header <boost/bitstream/iobmanip.hpp>

```
namespace boost {
  namespace bitstream {
    class setrepeat;
    class ignore;
    class aligng;
    ibstream & operator>>(ibstream &, setrepeat);
    ibstream & operator>>(ibstream &, ignore);
    ibstream & operator>>(ibstream &, aligng);
  }
}
```

Class setrepeat

boost::bitstream::setrepeat

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


class setrepeat {
public:
  // construct/copy/destruct
  setrepeat(size_t);

  // public member functions
  ibstream & operator()(ibstream &) const;
};
```

Description

This class represents the setrepeat bit-stream manipulator.

Note

This is a roundabout but necessary way of implementing manipulators that take parameters, such as the setprecision() manipulator.

`setrepeat` public construct/copy/destruct

1. ```
   setrepeat(size_t repeat);
   ```

   Constructor.

   Parameters: `repeat` Number of bit fields to extract to each subsequent integral container.

`setrepeat` public member functions

1. ```
   ibstream & operator()(ibstream & ibs) const;
   ```

   Overload for the () operator on this class.

   Parameters: `ibs` Reference to ibstream on lhs of >> operator.
   Returns: Reference to ibstream parameter.

Class ignore

boost::bitstream::ignore

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


class ignore {
public:
  // construct/copy/destruct
  ignore(size_t);

  // public member functions
  ibstream & operator()(ibstream &) const;
};
```

Description

This class represents the ignore bit-stream manipulator.

See Also:

Implementation note for setrepeat manipulator.

`ignore` public construct/copy/destruct

1. ```
   ignore(size_t bits);
   ```

   Constructor.

`ignore` public member functions

1. ```
   ibstream & operator()(ibstream & ibs) const;
   ```

   Overload for the () operator on this class.

   Parameters: `ibs` Reference to ibstream on lhs of >> operator.
   Returns: Reference to ibstream parameter.

Class aligng

boost::bitstream::aligng

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


class aligng {
public:
  // construct/copy/destruct
  aligng(size_t);

  // public member functions
  ibstream & operator()(ibstream &) const;
};
```

Description

This class represents the aligng bit-stream manipulator.

See Also:

Implementation note for setrepeat manipulator.
Examples:

```
// Advance get pointer to next nibble.
static const bitset<2> version(0x2);
bitset<4> csrcCount;
bool marker;
bitset<7> payloadType;
DWORD timestamp, ssrcIdentifier;
ibitstream bin(rtpHeader);
bin >> version >> aligng(4) >> csrcCount >> marker >> payloadType;
bin >> timestamp >> ssrcIdentifier;
```

```
// Advance get pointer to next word.
bitset<16> sequenceNumber;
ibitstream(rtpHeader).aligng(sizeof(WORD) * CHAR_BIT) >> sequenceNumber;
```

```
// Advance get pointer to next double word.
bool b;
static const bitset<2> version(0x2);
bitset<4> csrcCount;
bitset<7> payloadType;
DWORD timestamp;
ibitstream bin(rtpHeader);
bin >> version >> b >> b >> csrcCount >> b >> payloadType
    >> aligng(32) >> timeStamp;
```

`aligng` public construct/copy/destruct

1.
```
aligng(size_t bits);
```

Constructor.

Parameters: `bits` Number of bits at which to align the get pointer.

`aligng` public member functions

1.
```
ibstream & operator()(ibstream & ibs) const;
```

Overload for the () operator on this class.

Parameters: `ibs` Reference to ibstream on lhs of >> operator.
Returns: Reference to ibstream parameter.

Function operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


ibstream & operator>>(ibstream & ibs, setrepeat repeat);
```

Description

Manipulator for ibstream that sets repeat count for subsequent container extractions.

Parameters: `ibs` Reference to ibstream on left-hand side of operator.
`repeat` Instance of setrepeat class.
Returns: Reference to ibstream parameter.

Function operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


ibstream & operator>>(ibstream & ibs, ignore skip);
```

Description

Manipulator for ibstream that ignores bits.

Parameters: `ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

Function operator>>

boost::bitstream::operator>>

Synopsis

```
// In header: <boost/bitstream/iobmanip.hpp>


ibstream & operator>>(ibstream & ibs, aligng align);
```

Description

Manipulator for ibstream that aligns the get pointer in the input stream.

Parameters: `align` Instance of aligng class.
`ibs` Reference to ibstream on left-hand side of operator.
Returns: Reference to ibstream parameter.

Class Index

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

X

Typedef Index

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

X

Function Index

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

X

Macro Index

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

X

Index

A

B

C

D

E

F

G

H

I

O

P

Q

R

S

T

U

V

X